# Unifying lower bounds for algebraic machines, semantically

Thomas Seiller, Luc Pellissier, Ulysse Léchine

# Unifying lower bounds for algebraic machines, semantically

Thomas Seiller[a,1,*], Luc Pellissier[b,2], Ulysse Léchine[c]

[a] *CNRS, LIPN – UMR 7030 CNRS & University of Paris 13, 99 avenue Jean-Baptiste Clément, 93430, Villetaneuse, France*
[b] *University Paris Est Creteil, LACL, Faculté des Sciences et Technologie, 61 avenue du Général de Gaulle, 94010, Creteil, France*
[c] *Université Sorbonne Paris Nord, LIPN – UMR 7030 CNRS & University of Paris 13, 99 avenue Jean-Baptiste Clément, 93430, Villetaneuse, France*

**Abstract**

This paper presents a new abstract method for proving lower bounds in computational complexity. Based on the notion of topological and measurable entropy for dynamical systems, it is shown to generalise three previous lower bounds results from the literature in algebraic complexity. We use it to prove that `maxflow`, a PTIME complete problem, is not computable in polylogarithmic time on parallel random access machines (PRAMs) working with real numbers. This improves, albeit slightly, on a result of Mulmuley since the class of machines considered extends the class "PRAMs without bit operations", making more precise the relationship between Mulmuley's result and similar lower bounds on real PRAMs.

More importantly, we show our method captures previous lower bounds results from the literature, thus providing a unifying framework for "topological" proofs of lower bounds: Steele and Yao's lower bounds for algebraic decision trees [1], Ben-Or's lower bounds for algebraic computation trees [2], Cucker's proof that NC is not equal to PTIME in the real case [3], and Mulmuley's lower bounds for "PRAMs without bit operations" [4].

*Corresponding author

*Email addresses:* `thomas.seiller@cnrs.fr` (Thomas Seiller), `luc.pellissier@lacl.fr` (Luc Pellissier), `lechine@lipn.fr` (Ulysse Léchine)

[1] T. Seiller was partially supported by the European Commission Horizon 2020 programme Marie Skłodowska-Curie Individual Fellowship (H2020-MSCA-IF-2014) project 659920 - ReACT, the INS2I grants BIGRE and LOBE, the Ile-de-France DIM RFSI Exploratory project Exploratory project COHOP, and the ANR-22-CE48-0003-01 project DYSCO.

[2] L. Pellissier was partially supported by ANR-14-CE25-0005 project ELICA and ANII project "Realizabilidad, forcing y computación cuántica" FCE_1_2014_1_104800.

**Contents**

**1 Introduction** **2**

**2 Contents of the paper** **7**

**3 Programs as Dynamical systems** **15**

**4 Algebraic models of computations as AMCs** **20**

**5 Entropy and Cells** **28**

**6 First lower bounds** **33**

**7 Refining the method** **36**

**8 Recovering Ben Or and Cucker's theorems** **41**

**9 Algebraic surfaces for an optimization problem** **43**

**10 Improving Mulmuley's result** **49**

## 1. Introduction

### *1.1. Computational Complexity*

The field of computational complexity was initiated soon after the conception of the first computers. While theoretical results had already established a definition of the notion of "computable function" on the set of natural numbers, it became quickly apparent that computable did not mean practical, as many functions considered computable could not be computed within a reasonable time.

The first complexity class defined was that of *feasible functions* [5, 6, 7], which is now known as PTIME: the set of polynomial time computable functions, i.e. functions $f$ for which there exists a polynomial $p$ and a (usually Turing) machine $M$ computing $f$ whose running time on an input $n$ is bounded by $p(n)$. This class, apart from being the first ever complexity class to appear in the literature, is arguably the most important one in computer science. Many fundamental problems concern its relation to other classes, such as knowing wheter NPTIME, the extension of PTIME if one allows for non-deterministic machines in the above definition, is equal to PTIME. These problems, however, are still open.

Beyond the relationship between PTIME and other classes, the general question of *classifying* the complexity classes became one of the main objectives of the field, and a number of important results were obtained within the first years.

*1.2. Separation, Lower bounds and Barriers*

As part of the classification problem, complexity theory has traditionally been concerned with proving *separation results*. Among the numerous open separation problems lies the much advertised PTIME vs. NPTIME problem of showing that some problems considered hard to solve but efficient to verify do not have a polynomial time algorithm solving them.

Proving that two classes $B \subset A$ are not equal can be reduced to finding lower bounds for problems in $A$: by proving that certain problems cannot be solved with less than certain resources on a specific model of computation, one can show that two classes are not equal. Conversely, proving a separation result $B \subsetneq A$ provides a lower bound for the problems that are $A$-*complete* [8] – i.e. problems that are in some way *universal* for the class $A$.

The proven lower bound results are however very few, and most separation problems remain as generally accepted conjectures. For instance, a proof that the class of non-deterministic exponential problems is not included in what is thought of as a very small class of circuits was not achieved until very recently [9].

The failure of most techniques of proof has been studied in itself, which lead to the proof of the existence of negative results that are commonly called *barriers*. Altogether, these results show that all proof methods we know are ineffective with respect to proving interesting lower bounds. Indeed, there are three barriers: relativisation [10], natural proofs [11] and algebrization [12, 13], and every known proof method hits at least one of them, which shows the need for new methods[3]. However, to this day, only one research program aimed at proving new separation results is commonly believed to have the ability to bypass all barriers: Mulmuley and Sohoni's Geometric Complexity Theory (GCT) program [14].

*1.3. Algebraic models and Geometric Complexity Theory*

Geometric Complexity Theory (GCT) is widely considered to be a promising research program that might lead to interesting results. It is also widely believed to necessitate new and extremely sophisticated pieces of mathematics in order to achieve its goal. The research program aims in the long run to provide new techniques for answering the PTIME versus NPTIME problem, focussing first on solving the VP versus VNP problem by showing that certain algebraic surfaces (representing the permanent and the determinant) cannot be embedded one into the other. Although this program has lead to interesting developments in pure mathematics, it has not enhanced our understanding of complexity lower bounds for the time being (actually, even for Mulmuley himself, such understanding will not be achieved in our lifetimes [15]).

Intuitively, this program develops a proof method for proving lower bounds in *algebraic complexity* based on algebraic geometry [14]: separation of the Valiant

[3] In the words of S. Aaronson and A. Wigderson [12], "We speculate that going beyond this limit [algebrization] will require fundamentally new methods."

complexity classes VP and VNP could be deduced from the impossibility of embedding an algebraic variety $\mathcal{P}$ defined from the permanent into an algebraic variety $\mathcal{D}$ defined from the determinant (with constraints on the dimensions). Two main approaches were proposed using representation theory and exploiting the decomposition of representations of varieties into irreducible components:

- *occurence obstructions* aims to exhibit an irreducible representation of $G = GL_k\mathbf{C}$ which occurs as a $G$-subrepresentation in the coordinate ring of $\mathcal{P}$ but not occurring as a $G$-subrepresentation in the coordinate ring of $\mathcal{D}$, while
- *multiplicity obstructions* an irreducible representation of $G = GL_k\mathbf{C}$ which occurs as a $G$-subrepresentation in both the coordinate ring of $\mathcal{P}$ and in the coordinate ring of $\mathcal{D}$, but whose multiplicity of occurence in the coordinate ring of $\mathcal{P}$ is strictly greater than its multiplicity of occurence in the coordinate ring of $\mathcal{D}$.

Obviously, the first approach is easier, as a particular case of the second. Recently, some negative results [16] have shown this easiest path proposed by GCT fails. Some positive results on a toy model were however obtained regarding multiplicity obstructions [17]: although the obtained results are not new, they use the multiplicity obstruction method and are considered a proof-of-concept of the approach.

The GCT program was inspired, according to its creators, by a lower bound result obtained by Mulmuley [4] for "PRAMs without bit operations", a result we strengthen in the present work.

### *1.4. Lower bounds for* PRAMs *without bit operations*

Mulmuley showed in 1999 [4] that a notion of machine introduced under the name "PRAMs without bit operations" does not compute `maxflow` in polylogarithmic time. This notion of machine, quite exotic at first sight, corresponds to an algebraic variant of PRAMs, where registers contain integers and individual processors are allowed to perform sums, subtractions and products of integers. It is argued by Mulmuley that this notion of machine provides an expressive model of computation, able to compute some non trivial problems in NC such as Neff's algorithm for computing approximate roots of polynomials [18]. Mulmuley's result is understood as a big step forward in the quest for a proof that PTIME and NC are not equal. However, the result was not strengthened or reused in the last 20 years, and remained the strongest known lower bound result in this line of enquiry.

The `maxflow` problem is quite interesting as it is known to be in PTIME (by reduction to linear programming, or the Ford-Fulkerson algorithm [19]). In fact, it is PTIME complete [20], and (obviously) there are currently no known efficiently parallel algorithm solving it. This lower bound proof, despite being the main inspiration of the well-known GCT research program, remains seldom cited and has not led to variations applied to other problems.

*Remark.* In fact, the article by Mulmuley shows lower bounds for several problems, and not only for the `maxflow` problem. While this paper focusses on `maxflow`, this is a choice motivated by the particular importance of the latter (`maxflow` being PTIME-complete [**?** ]). However, the lower bounds obtained by Mulmuley for other problems can be obtained and strengthened in a similar way using our technique.

*1.5. Contributions.*

The first contribution of this work is a strengthening of Mulmuley's lower bounds result for machines working on real numbers. Indeed, while the latter proves that `maxflow` is not computable in polylogarithmic time in a variant of *arithmetic* PRAMs, i.e. working with integers, the proof uses in an essential way techniques from real algebraic geometry. We will explain how this result is in fact a consequence of our main technical lemma, i.e. follows from lower bounds for *algebraic* PRAMs, that is machines working on the reals. Indeed, we show that division-free polylogarithmic algebraic PRAMs compute the same sets of integers as division-free polylogarithmic arithmetic PRAMs (Proposition 71).

We then show that `maxflow` is in fact not computable in polylogarithmic time in a more expressive model of algebraic PRAMs, in which processors are allowed to perform arbitrary divisions and arbitrary roots in addition to the basic operations allowed in Mulmuley's case (addition, subtraction, multiplication). We then explain how this more general result fails to lift to the arithmetic case, pinpointing to the precise reason it does: that euclidean division (in fact division by 2) is not computable in polylogarithmic time by algebraic PRAMs. This leads us to prove that the corresponding class, which contains the problems computable by Mulmuley's notion of PRAMs, is in fact strictly contained in NC. This leads to the main technical result of our paper:

**Theorem 1.** *Let $N$ be a natural number and $M$ be a real-valued* PRAM *with at most $2^{O((\log N)^c)}$ processors, where $c$ is any positive integer.*

*Then $M$ does not compute euclidean division by 2 on inputs of length $N$ in $O((\log N)^c)$ steps.*

This result improves the result shown by Mulmuley, since the class of machines he considers[4] is strictly contained in the class of machines considered in the statement.

The second, and main, contribution of the paper is the proof method itself, which is based on *dynamic semantics* for programs by means of *graphings*, a notion introduced in ergodic theory and recently used to define models of linear logic by the first author [21, 22, 23, 24]. The dual nature of graphings, both continuous and discrete, is essential in the present work, as it enables invariants from continuous mathematics, in particular the notion of *topological entropy* for dynamical systems, while the finite representability of graphings is used in the

[4]As mentioned above, Mulmuley considers integer-valued PRAMs, but this class computes exactly the restrictions of sets decided by real-valued PRAMs to integral points.

key lemma (as the number of *edges* appears in the upper bounds of Lemma 55 and Lemma 5).

In particular, we show how this proof method captures known lower bounds and separation results in algebraic models of computation, namely Steele and Yao's lower bounds for algebraic decision trees [1], Ben-Or's lower bounds on algebraic computation trees [2], Cucker's proof that $\mathrm{NC}_{\mathbf{R}}$ is not equal to $\mathrm{PTIME}_{\mathbf{R}}$ [3] (i.e. answering the NC vs PTIME problem for computation over the real numbers).

### *1.6. A more detailed view of the proof method*

One of the key ingredients in the proof is the representation of programs as graphings, and *quantitative soundness* results. We refer to the next sections for a formal statement, and we only provide an intuitive explanation for the moment. Since a program $P$ is represented as a graphing $[\![P]\!]$, which is in some way a dynamical system, the computation $P(a)$ on a given input $a$ is represented as a sequence of values $[\![a]\!], [\![P]\!]([\![a]\!]), [\![P]\!]^2([\![a]\!]), \ldots$ – an orbit for the dynamical system. Quantitative soundness states that not only $[\![P]\!]$ computes exactly the same function (where computation is understood as convergence to a stationary value) as the original program $P$, but it does so with a constant time overhead, i.e. if $P(a)$ terminates on a value $b$ in time $k$, then $[\![P]\!]^{Ck}([\![a]\!]) = [\![b]\!]$, where $C$ is a constant fixed once and for all for the model of computation.

The second ingredient is the dual nature of graphings, both continuous and discrete objects. Indeed, a graphing *representative* is a graph-like structure whose edges are represented as continuous maps, i.e. a finite representation of a (partial) continuous dynamical system. Given a graphing, we define its *kth cell decomposition*, which separates the configuration space into cells such that two inputs in the same cell are indistinguishable in $k$ steps, i.e. the graphing's computational traces on both inputs are equal. We can then use both the finiteness of the graphing representatives and the *topological entropy* of the associated dynamical system to provide upper bounds on the size of a further refinement of this geometric object, namely the *k-th entropic co-tree* of a graphing – a kind of final approximation of the graphing by a computational tree: intuitively, the $k$-th entropic co-tree is a computational tree that mimicks the behaviour of the graphing for the $k$ final steps of computation.

As we deal with algebraic models of computation, this implies a bound on the representation of the $k$th cell decomposition as a semi-algebraic variety. In other words, the $k$th cell decomposition is defined by polynomial in·equalities and we provide bounds on the number and degree of the involved polynomials. The corresponding statement is the main technical result of this paper (Lemma 5).

This lemma can then be used to obtain lower bounds results that we now detail. Precise definitions of the complexity classes involved can be found in Section 4.

*Computational trees..* Using the Milnor-Oleĭnik-Petrovskii-Thom theorem [25, 26, 27] to bound the number of connected components of the $k$th cell decomposition,

we then recover the lower bounds of Steele and Yao on algebraic decision trees, and the refined result of Ben-Or providing lower bounds for algebraic computation trees. In fact, we even slightly generalise Ben-Or's result as we obtain lower bounds for the model extended with arbitrary roots, while Ben-Or's original paper only considered square roots.

*Cucker's result..* A different argument based on invariant polynomials provides a proof of Cucker's result that $\mathrm{NC}_{\mathbf{R}} \neq \mathrm{PTIME}_{\mathbf{R}}$ by showing that a given polynomial that belongs to $\mathrm{PTIME}_{\mathbf{R}}$ cannot be computed within $\mathrm{NC}_{\mathbf{R}}$. In fact, our main technical result also shows that euclidean division by 2 cannot be computed by algebraic circuits. We present a direct proof of this result inspired from the general entropic co-trees approach (Section 10.2). This result is also a direct corollary of theorem 1.

*Mulmuley's result..* Lastly, following Mulmuley's geometric representation of the `maxflow` problem, we are able to strenghthen his celebrated result to obtain lower bounds on the size (depth) of a PRAM over the reals computing this problem. While this result holds for real-valued PRAMs, we explain how it fails to extend to integer-valued machines, leading to a further strengthening of the result.

*Euclidean division..* We then explain how to further strengthen the result by showing that euclidean division cannot be computed in this algebraic PRAMs model in polylogarithmic time. More precisely, we show that euclidean division by 2 cannot be computed, leading to the result. Intuitively, this is due to the exponential number of breakpoints in the geometric representation of euclidean division. This extends the direct proof that division by 2 cannot be computed by algebraic circuits obtained earlier, using the entropic co-tree method introduced in this paper.

## 2. Contents of the paper

### *2.1. Computation models as graphings.*

The present work reports on the first investigations into how the interpretation of programs as graphings (generalised dynamical systems) could shed a new light on proofs of lower bounds. This interpretation of programs rely on two ingredients:

- the interpretation of models of computation as monoid actions. In our setting, we view the computational principles of a computational model as elements that act on a configuration space. As these actions can be composed, but are not necessarily reversible, it is natural to interpret them as a monoid acting on a configuration space.
- the realization of programs as graphings. We abstract programs as graphs whose vertices are subspaces of the product of the configuration space and the control states and edges are labelled by elements of the acting monoid, acting on subspaces of vertices.

Let us illustrate how monoid actions and graphings formalise the notions of model of computation and program. We consider Turing machines, and mathematically represent the model as the following monoid action. We consider the space of configurations $X \times S$, where $X = \{\star, 0, 1\}^{|\mathbf{Z}|}$ of $\mathbf{Z}$-indexed sequences of symbols $\star, 0, 1$ that are almost always equal to $\star$ and $S$ is a finite set of *control states*. A given point in this configuration space, extended with a chosen *control state*, describes a configuration of a Turing machine. Now, instructions present in the model give rise to maps from $X$ to $X$: for instance moving the working head to the right can be represented as $\texttt{right} : X \to X$, $(a_i)_{i\in\mathbf{Z}} \mapsto (a_{i+1})_{i\in\mathbf{Z}}$, that is the usual shift operator. The set of instructions then generates a monoid action $M \curvearrowright X$, or equivalently a monoid of endomorphisms of $X$, namely the monoid generated by the maps induced by the instructions. A graphing is then a collection of *edges* consisting of a source (a subspace of $X \times S$) and a realiser (an element of the monoid $M$ and a target state in $S$). The instruction "if in control state $a$ and the head is reading a 0 or a 1, move to the right and move to control state $b$" is then represented as an edge of source the subspace $\{(a_i)_{i\in\mathbf{Z}} \in X \mid a_0 \neq \star\} \times \{a\}$ and realised by the map $\texttt{right} \times (a \mapsto b)$.

The basic intuitions here can be summarised by the following slogan: "Computation, as a dynamical process, can be modelled as a dynamical system". Of course, the above affirmation cannot be true of all computational processes; for instance the traditional notion of dynamical system is deterministic. In practice, one works with a generalisation of dynamical systems named *graphings*. Introduced in ergodic theory [28, 29, 30], graphings were recently used in theoretical computer science to define realisability models of linear logic [22, 24, 21], in which they are shown to model non-deterministic and probabilistic computation [23, 31].

To do so, we consider that a computation model is given by a set of generators (representing basic instructions) and its actions on a space (representing the configuration space). So, in other words, we define a computation model as an action of a monoid (presented by its generators and relations) on a space $\alpha : M \curvearrowright \mathbf{X}$. This action can then be specified to be continuous, mesurable, etc. depending on the properties we are interested in.

A program in such a model of computation is then viewed as a *graphing*: a graph whose vertices are subspaces of the configuration space and edges are generators of the monoid: in this way, the partiality of certain operations and branching are both allowed. This point of view is very general, as it can allow to study, as special model of computations, models that can be discrete or continuous, algebraic, rewriting-based, etc.

A point in the configuration space is sent (potentially non-deterministically) through the graphing to other points: they constitute the *orbit* of the point under the graphing. This orbit can be eventually stationnary, meaning that the computation has reached a result, or have any kind of complex behavior. In any case, the study of the orbits of a graphing contain a lot of information on the graphing as will be clear when studying entropy.

*2.2. The general algebraic model*

We are able to introduce PRAMs acting over integers or real numbers in this setting. They can be described as having a finite number of processors, each having access to a private memory on top of the shared memory, and able to perform the operations $+, -, \times$ – and possibly division and root operations (depending on the models considered) –, as well as branching and indirect addressing. Interestingly, we can represent these machines in the graphings framework in two steps: first, by defining the (sequential) RAM model, with just one processor; and then by performing an algebraic operation corresponding to parallelisation on the corresponding monoid action.

The RAM model on integers has, as configuration space $\mathbf{Z}^{|\mathbf{Z}|}$ an infinite array of cells each containing a real number, and the operations are the usual ones. The equivalent model on the reals has as configuration space $\mathbf{R}^{|\mathbf{R}|}$; the indexing by real numbers eases the definition in the presence of indirect addressing, even though in practise only a finite number of registers can be accessed and/or modified in a finite computation.

Parallel computation is thus modelled *per se*, at the level of the model of computation. As usual, one is bound to chose a mode of interaction between the different processes when dealing with shared memory. We will consider here only the case of *Concurrent Read Exclusive Write* (CREW), i.e. all processes can read the shared memory concurrently, but if several processes try to write in the shared memory only the process with the smallest index is allowed to do so.

The heart of our approach of parallelism is based on commutation. Among all the instructions, the ones affecting only the private memory of distinct processors can commute, while it is not the case of two instructions affecting the central memory. We do so by considering a notion of product for monoids (definition 28) that generalizes both the direct product and the free product: we specify, through a conflict relation, which of the generators can and can not commute, allowing us to build a monoid representing the simultaneous action.

To ease the presentation, and since many different algebraic models are considered in the paper, we introduce a very general abstract model of computation, e.g. the monoid action $\alpha_{\mathbf{R}\text{full}}$ for machines computing on the reals (Definition 13). We then show that all notions of machines considered in the present paper can be adequately represented by considering restrictions of $\alpha_{\mathbf{R}\text{full}}$. The main technical lemma being proved for the most general action $\alpha_{\mathbf{R}\text{full}}$, it then applies naturally to all those models: algebraic computational trees, algebraic circuits, PRAMs over the reals, etc.

*2.3. Entropy*

We fix an action $\alpha : M \curvearrowright \mathbf{X}$ for the following discussion (it can be thought of as the most general action $\alpha_{\mathbf{R}\text{full}}$ that captures all algebraic models of interest in this work). One important aspect of the representation of abstract programs as graphings is that restrictions of graphings correspond to known notions from mathematics. In a very natural way, a deterministic $\alpha$-graphing defines a partial dynamical system. Conversely, a partial dynamical system whose graph is

contained in the *measured preorder* $\{(x,y) \in \mathbf{X}^2 \mid \exists m \in M, \alpha(m)(x) = y\}$ [32] can be associated to an $\alpha$-graphing.

The study of deterministic models of computations can thus profit from the methods of the theory of dynamical systems. In particular, the methods employed in this paper relate to the classical notion of *topological entropy*. The topological entropy of a dynamical system is a value representing the average exponential growth rate of the number of orbit segments distinguishable with a finite (but arbitrarily fine) precision. The definition is based on the notion of open covers: for each finite open cover $\mathcal{C}$, one can compute the entropy of a map w.r.t. $\mathcal{C}$, and the entropy of the map is then the supremum of these values when $\mathcal{C}$ ranges over the set of all finite covers. As we are considering graphings and those correspond to partial maps, we explain how the techniques adapt to this more general setting and define the entropy $h(G, \mathcal{C})$ of a graphing $G$ w.r.t. a cover $\mathcal{C}$, as well as the topological entropy $h(G)$ defined as the supremum of the values $h(G, \mathcal{C})$ where $\mathcal{C}$ ranges over all finite open covers.

While the precise results described in this paper use the entropy $h_0(G)$ w.r.t. a specific cover (similar bounds could be obtained from the topological entropy, but would lack precision), the authors believe entropy could play a much more prominent role in future proofs of lower bounds. Indeed, while $h_0(G)$ somehow quantifies over one aspect of the computation, namely the branchings, the topological entropy computed by considering all possible covers provides a much more precise picture of the dynamics involved. In particular, it provides information about the computational principles described by the action; this information may lead to more precise bounds based on how some principles are much more complex than some others, providing some lower bounds on possible simulations of the former with the latter.

### *2.4. Known lower bounds and entropy*

All the while only the entropy w.r.t. a given cover will be essential in this work, the overall techniques related to topological entropy provide a much clearer picture of the techniques. We first use the following lemma bounding the $k$-cell decomposition of a given graphing.

**Proposition 2.** *Let $G$ be a deterministic graphing. We consider the* state cover entropy $h_0([G]) = \lim_{n\to\infty} H^n_{\mathbf{X}}([G], \mathcal{S})$ *where $\mathbf{S}$ is the state cover. The cardinality of the $k$-th cell decomposition of $\mathbf{X}$ w.r.t. $G$, as a function $c(k)$ of $k$, is asymptotically bounded by $g(k) = 2^{k.h_0([G])}$, i.e. $c(k) = O(g(k))$.*

This lemma can be used to derive known lower bounds from the literature, namely Steele and Yao's result on *algebraic decision trees*. Algebraic decision trees are finite ternary trees describing a program deciding a subset of $\mathbf{R}^n$: each node verifies whether a chosen polynomialtakes a positive, negative, or null value at the point considered. A $d$-th order algebraic decision tree is an algebraic decision tree in which all polynomials are of degree bounded by $d$.

In a very natural manner, an algebraic decision tree can be represented as a $\iota$-graphing, where $\iota$ is the trivial action on the space $\mathbf{R}^n$. We use entropy to

provide a bound on the number of connected components of subsets decided by $\iota$-graphings. These bounds are obtained by combining a bound in terms of entropy and a variant of the Milnor-Oleĭnik-Petrovskii-Thom theorem due to Ben-Or. The latter, which we recall below, bounds the number of connected components of a semi-algebraic set in terms of the number of polynomial inequalities, their maximal degree, and the dimension of the space considered. These bounds then lead to the lower bounds on algebraic decision trees obtained by Steele and Yao.

**Corollary 3** (Steele and Yao [1])**.** *A d-th order algebraic decision tree deciding a subset $W \subseteq \mathbf{R}^n$ with $N$ connected components has height $\Omega(\log N)$.*

Mulmuley's original result on PRAMs without bit operations can also be obtained as a corollary of this bound, associated with a more involved geometric argument. We detail and rephrase the latter (in a way that will allow us to reuse it in the last section) in section 9. Combining the results of this section and the previous lemma about the $k$-cell decomposition, one obtains Mulmuley's theorem.

**Corollary 4** (Mulmuley [4])**.** *Let $M$ be a* PRAM *without bit operations, with at most $2^{O((\log N)^c)}$ processors, where $N$ is the length of the inputs and $c$ any positive integer.*

*Then $M$ does not decide* `maxflow` *in $O((\log N)^c)$ steps.*

*2.5. Entropic co-tree and the technical lemma*

The above result of Steele and Yao adapts in a straightforward manner to a notion of algebraic computation trees describing the construction of the polynomials to be tested by mean of multiplications and additions of the coordinates. As shown above, this result uses techniques quite similar to that of Mulmuley's lower bounds for the model of PRAMs *without bit operations*. The authors also quickly realised the techniques is similar to that used by Cucker in proving that $\mathrm{NC}_{\mathbf{R}} \neq \mathrm{PTIME}_{\mathbf{R}}$ [3].

It turns out a refinement of Steele and Yao's method was quickly obtained by Ben-Or [2] so as to obtain a similar result for an extended notion of algebraic computation trees allowing for computing divisions and taking square roots. We adapt Ben-Or techniques within the framework of graphings, in order to apply this refined approach to Mulmuley's framework, leading to a strengthened lower bounds result.

This techniques refines the bounds on the $k$th-cell decomposition explained above by considering *entropic co-trees*. They are defined in a similar way as the $k$-th cell decomposition but further track the different instructions involved. The result is a directed graph in the form of a tree with all edges pointing toward the root (hence the name of *co-tree*). This tree can be understood as a final approximation of the graphing as a computational tree: it is a computational tree whose behaviour mimicks that of the graphing in the last computation steps leading to acceptation or rejection. At each fixed depth, the set of vertices of the co-tree refine the $k$-th cell decomposition explained above. The additional information related to the instructions realising the edges of the co-tree can be

used to derive a set of polynomial in·equalities whose total degree can be bounded by the depth, the state-cover entropy $h_0$, the algebraic degree (Definition 56) – the maximal number of instructions used in a single edge of the graphing –, and the root degree $\sqrt[\partial]{G}$ – the largest integer $d$ such that the $d$-th root instruction appears in the graphing.

This leads to the following technical lemma, from which most of the subsequent lower bound results will be obtained.

**Lemma 5.** *Let $G$ be a CREW$^p(\alpha_{\mathbf{R}\text{full}})$-computational graphing representative, $\mathrm{Seq}_k(E)$ the set of length $k$ sequences of edges in $G$, and $D$ its algebraic degree. Suppose $G$ computes the membership problem for $W \subseteq \mathbf{R}^n$ in $k$ steps, i.e. for each element of $\mathbf{R}^n$, $\pi_{\mathbf{S}}(G^k(x)) = \top$ if and only if $x \in W$. Then $W$ is a semi-algebraic set defined by at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ systems of $pkD$ equations of degree at most $\max(2, \sqrt[\partial]{G})$ and involving at most $pD(k+n)$ variables.*

As a first corollary of this theorem, we obtain a generalisation of Ben-Or's result (Theorem 57). It follows from the above theorem and the Milnor-Oleĭnik-Petrovskii-Thom theorem bounding the number of connected components of a semi-algebraic set. A corollary of this general result is Ben-Or's original lower bounds for Algebraic Computational Trees.

**Corollary 6** ([2, Theorem 5])**.** *Let $W \subseteq \mathbf{R}^n$ be any set, and let $N$ be the maximum of the number of connected components of $W$ and $\mathbf{R}^n \setminus W$. An algebraic computation tree computing the membership problem for $W$ has height $\Omega(\log N)$.*

The above lemma is then applied to obtain Cucker's theorem that $\mathrm{NC}_{\mathbf{R}} \neq \mathrm{PTIME}_{\mathbf{R}}$.

**Corollary 7.** *No algebraic circuit of depth $k = \log^i n$ and size $kp$ compute $\mathfrak{F}$er:*

$$\{x \in \mathbf{R}^\omega \mid |x| = n \Rightarrow x_1^{2^n} + x_2^{2^n} = 1\}.$$

Finally, the lemma can be used to obtain a first strengthening of Mulmuley's lower bound (Theorem 8). As explained above, Mulmuley's result for PRAMs without bit operations (working on integers) corresponds to lower bounds on division-free algebraic – i.e. working on the reals – PRAMs. By using the above technical lemma, we are able to extends the lower bounds for computing `maxflow` to algebraic PRAMs with division and arbitrary roots.

Before stating this result and explaining how we are able to further generalise it, we sketch Mulmuley's geometric method. While the results here are not new, the authors' contribution is that of reformulation. In particular, we reorganise the geometric part of the proof as the combination of two results: a previous result of Murty and Carstensen showing the existence of an exponential linear parametrization of `maxflow`, and a general geometric statement (Theorem 68). This reformulation will be used in later sections to show that a problem easier than `maxflow` cannot be computed in the algebraic PRAMs model.

*2.6. Mulmuley's geometrization*

Contrarily to Ben-Or's model, the PRAM machines do not decide sets of reals but of integers, making the use of algebraico-geometric results to uncover their geometry much less obvious. The mechanisms of Mulmuley's proof rely on twin geometrizations: one of a special optimization problem that can be represented by a surface in $\mathbf{R}^3$ (subsections 9.1 and 9.2), the other one by building explicitly, given a PRAM, a set of algebraic surfaces such that the points accepted by the machine are exactly the integer points enclosed by the set of surfaces.

That second part can be abstracted as a relation between the sets decided by PRAMs without bit operations and division-free algebraic PRAMs. In other words, the set of integral points accepted by a PRAM without bit operations coincides with the integral points lying in the set decided by a corresponding algebraic PRAM. This is stated as (Proposition 71).

Finally, the proof is concluded by a purely geometrical theorem (Theorem 68). We would like to stress here that this separation in three movement, with a geometrical tour-de-force, is not explicit in the original article. We nonetheless believe it greatly improves the exposition (on top of allowing for a stregthening of the results). This geometric theorem expresses a tension between the two geometrizations. Our work focuses here only on the construction of a set of algebraic surfaces representing the computation of a PRAM; the remaining part of our proof follows Mulmuley's original technique closely.

*Building surfaces.* The first step in Mulmuley's proof is to use the parametric complexity results of [33] to represent an instance of the decision problem associated to `maxflow` so that it induces naturally a partition of $\mathbf{Z}^3$ that can then be represented by a surface.

The second step is to represent any partition of $\mathbf{Z}^3$ induced by the run of a machine by a set of surfaces in $\mathbf{R}^3$, in order to be able to use geometric methods.

Let $K$ be a compact of $\mathbf{R}^3$ and $P = (P_1, \ldots, P_m)$ be a partition of $\mathbf{Z}^3 \cap K$. $P$ can be extended to a partition of the whole of $K$ in a number of ways, as pictured in Fig. 1. In particular, $P$ can always be extended to a partition $P_{\text{alg}}$ (resp. $P_{\text{smooth},}$, $P_{\text{ana}}$) of $K$ such that all the cells are compact, and the boundaries of the cells are all algebraic (resp. smooth, analytic) surfaces.

In general, such surfaces have no reason to be easy to compute and the more they are endowed with structure, the more complicated to compute they are to be. In the specific case of PRAMs, the decomposition can naturally be represented with algebraic surfaces whose degree is bounded. This choice of representation might not hold for any other model of computation, for which it might be more interesting to consider surfaces of a different kind.

This set of algebraic surfaces is here built just as in our description of Ben-Or's result using the entropic co-tree: we construct the co-tree approximating the computation of a specific PRAM and build along the branches of this co-tree a system of polynomial equations on a larger space than the space of variables actually used by the machine.This system of integer polynomials of bounded degree then defines surfaces exactly matching our needs, since the number of

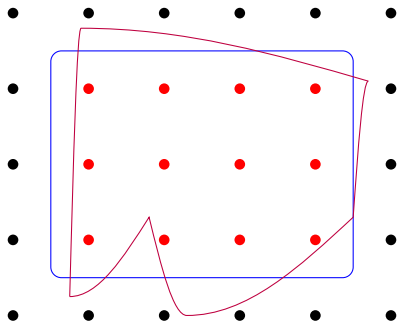


Figure 1: Two curves that define the same partition of $\mathbf{Z}^2$

varieties and their maximal degrees are bounded using the technical lemma described above.

*2.7. A first improvement on Mulmuley's lower bounds*

Interestingly, this allows us to derive from the technical lemma a proof that algebraic PRAMs, with division and arbitrary roots instructions on top of the instructions allowed in the PRAMs without bit operations model, cannot compute `maxflow` in polylogarithmic time. This is a first (arguably mild) improvement over Mulmuley's proof.

**Theorem 8.** *Let $N$ be a natural number and $M$ be a real-valued* PRAM *with at most $2^{O((\log N)^c)}$ processors, where $c$ is any positive integer.*

*Then $M$ does not decide* `maxflow` *on inputs of length $N$ in $O((\log N)^c)$ steps.*

Following Mulmuley's original method on division-free machines, one would then like to lift this result to arithmetic PRAMs, i.e. PRAMs over the integers with division (and possibly root operations). It turns out, however, that this cannot be done, a result not surprising since such machines can be shown to be equivalent to usual PRAMs.

More precisely, we can prove that algebraic PRAMs cannot compute euclidian division. We provide two proofs of this result. We first show that algebraic circuits (i.e. with division but no root operations) cannot compute euclidean division by 2 using a (new) direct proof method bounding the number of continuous pieces of the piece-wise continuous function computed by the circuit. This direct proof is a specific but more concrete instance of the general entropic co-tree approach, and illustrates well how the lower bounds are obtained. We further explain how to abstract the mathematical problem corresponding to generalising this bound to root operations, and illustrate the difficulty of doing so (even with only square roots).

In a second step, we use our technical lemma, together with the geometric result of Mulmuley, to provide lower bounds for computing the remainder modulo 2 in the general algebraic PRAMs model. This leads to the main technical result of this paper.

**Theorem 1.** *Let $N$ be a natural number and $M$ be a real-valued* PRAM *with at most $2^{O((\log N)^c)}$ processors, where $c$ is any positive integer.*

*Then $M$ does not compute the euclidean division modulo 2 on inputs of length $N$ in $O((\log N)^c)$ steps.*

This result shows a fundamental difference in expressive power between models with real-valued division and models allowed to compute the euclidean division. Indeed, this shows that a parallel model over the reals, even if allowed division and arbitrary roots, cannot compute euclidean division in polylogarithmic time.

### *2.8. Conclusion*

This work slightly strengthens Mulmuley's lower bounds on "PRAMs without bit operations". More importantly, it shows how the semantic techniques based on abstract models of computation and graphings can shed new light on several lower bound techniques. In particular, it establishes some relationship between the lower bounds and the notion of entropy which, although arguably still superficial in this work, could potentially become deeper and provide new insights and finer techniques.

Showing that the interpretation of programs as graphings can translate, and even refine, such strong lower bounds results is also important from another perspective. Indeed, the techniques of Ben-Or and Mulmuley (as well as other results of e.g. Cucker [3], Yao [34]) seem at first sight restricted to algebraic models of computation due to their use of the Milnor-Oleĭnik-Petrovskii-Thom theorem (or other geometric arguments) which holds only for real semi-algebraic sets. However, the second author's characterisations of Boolean complexity classes in terms of graphings acting on algebraic spaces [23] opens the possibility of using such algebraic methods to provide lower bounds for boolean models of computation.

## 3. Programs as Dynamical systems

### *3.1. Abstract models of computation and graphings*

We consider computations as a dynamical process, hence model them as a dynamical systems with two main components: a space $\mathbf{X}$ that abstracts the notion of configuration space (excluding control states) and a monoid acting on this space that represents the different operations allowed in the model of computation. Although the notion of *space* considered can vary (one could consider e.g. topological spaces, measure spaces, topological vector spaces), we restrict ourselves to topological spaces in this work.

**Definition 1.** An *abstract model of computation* (AMC) is a monoid action $\alpha : M \curvearrowright \mathbf{X}$, i.e. a monoid morphism from $M$ to the group of endomorphisms of $\mathbf{X}$. The monoid $M$ is often given by a set $G$ of generators and a set of relations R. We denote such an AMC as $\alpha : \langle G, \mathrm{R} \rangle \curvearrowright \mathbf{X}$.

Programs in an AMC $\alpha : \langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}$ is then defined as *graphings*, i.e. graphs whose vertices are subspaces of the space $\mathbf{X}$ (representing sets of configurations on which the program act in the same way) and edges are labelled by elements of $\mathrm{M}\langle G, \mathrm{R}\rangle$, together with a global control state. More precisely, we use here the notion of *topological graphings*[5] [22].

**Definition 2.** An $\alpha$-graphing representative $G$ w.r.t. a monoid action $\alpha : M \curvearrowright \mathbf{X}$ is defined as a set of *edges* $E^G$ together with a map that assigns to each element $e \in E^G$ a pair $(S_e^G, m_e^G)$ of a subspace $S_e^G$ of $\mathbf{X}$ – the *source* of $e$ – and an element $m_e^G \in M$ – the *realiser* of $e$.

While graphing representatives are convenient to manipulate, they do provide too much information about the programs. Indeed, if one is to study programs as dynamical systems, the focus should be on the *dynamics*, i.e. on how the object acts on the underlying space. The following notion of *refinement* captures this idea that the same dynamics may have different graph-like representations.

**Definition 3** (Refinement)**.** An $\alpha$-graphing representative $F$ is a refinement of an $\alpha$-graphing representative $G$, noted $F \leqslant G$, if there exists a partition $(E_e^F)_{e\in E^G}$ of $E^F$ such that $\forall e \in E^G$:

$$\begin{gathered}\forall f \neq f' \in E_e^F,\ S_f^F \triangle S_{f'}^F = \emptyset;\\ \forall f \in E_e^F,\ m_f^F = m_e^G.\end{gathered}$$

This induces an equivalence relation defined as

$$F \sim_{\mathrm{ref}} G \Leftrightarrow \exists H,\ H \leqslant F \wedge H \leqslant G.$$

The notion of *graphing* is therefore obtained by considering the quotient of the set of graphing representatives w.r.t. $\sim_{\mathrm{ref}}$. Intuitively, this corresponds to identifying graphings whose *actions on the underlying space are equal*.

**Definition 4.** An *$\alpha$-graphing* is an equivalence class of $\alpha$-graphing representatives w.r.t. the equivalence relation $\sim_{\mathrm{ref}}$.

We can now define the notion of abstract program. These are defined as graphings

**Definition 5.** Given an AMC $\alpha : M \curvearrowright \mathbf{X}$, an *$\alpha$-program* $A$ is a $\bar{\alpha}$-graphing $G^A$ w.r.t. the monoid action $\bar{\alpha} = \alpha \times \mathfrak{S}_k \curvearrowright \mathbf{X} \times \mathbf{S}^{\mathbf{A}}$, where $\mathbf{S}^{\mathbf{A}}$ is a finite set of *control states* of cardinality $k$ and $\mathfrak{S}_k$ is the group of permutations of $k$ elements.

Now, as a sanity check, we will show how the notion of graphing do capture the dynamics as expected. For this, we restrict to *deterministic graphings*, and show the notion relates to the usual notion of dynamical system.

[5]While "measured" graphings were already considered [22], the definition adapts in a straightforward manner to allow for other notions such as graphings over topological vector spaces – which would be objects akin to the notion of quiver used in representation theory.

**Definition 6.** An $\alpha$-graphing representative $G$ is deterministic if for all $x \in \mathbf{X}$ there is at most one $e \in E^G$ such that $x \in S_e^G$. An $\alpha$-graphing is *deterministic* if its representatives are deterministic. An abstract program is *deterministic* if its underlying graphing is deterministic.

**Lemma 7.** *There is a one-to-one correspondence between the set of deterministic graphings w.r.t. the action $M \curvearrowright \mathbf{X}$ and the set of partial dynamical systems $f : \mathbf{X} \hookrightarrow \mathbf{X}$ whose graph is contained in the preorder*[6] $\{(x, y) \mid \exists m \in M, \alpha(m)(x) = y\}$.

Lastly, we define some restrictions of $\alpha$-programs that will be important later. First, we will restrict the possible subspaces considered as sources of the edges, as unrestricted $\alpha$-programs could compute even undecidable problems by, e.g. encoding it into a subspace used as the source of an edge. Given an integer $k \in \omega$, we define the following subspaces of $\mathbf{R}^\omega$, for $\star \in \{>, \geqslant, =, \neq, \leqslant, <\}$:

$$\mathbf{R}^\omega_{k\star 0} = \{(x_1, \dots, x_k, \dots) \in \mathbf{R}^\omega \mid x_k \star 0\}.$$

**Definition 8** (Computational graphings)**.** Let $\alpha : \langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}$ be an AMC. A *computational $\alpha$-graphing* is an $\alpha$-graphing $T$ with distinguished states $\top$, $\bot$ which admits a finite representative such that each edge $e$ has its source equal to one among $\mathbf{R}^\omega$, $\mathbf{R}^\omega_{k\geqslant 0}$, $\mathbf{R}^\omega_{k\leqslant 0}$, $\mathbf{R}^\omega_{k>0}$, $\mathbf{R}^\omega_{k<0}$, $\mathbf{R}^\omega_{k=0}$, and $\mathbf{R}^\omega_{k\neq 0}$.

**Definition 9** (treeings)**.** Let $\alpha : \langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}$ be an AMC. An *$\alpha$-treeing* is an acyclic and finite $\alpha$-graphing, i.e. an $\alpha$-graphing $F$ for which there exists a finite $\alpha$-graphing representative $T$ whose set of control states $\mathbf{S^T} = \{0, \dots, s\}$ can be endowed with an order $<$ such that every edge of $T$ is state-increasing, i.e. for each edge $e$ of source $S_e$, for all $x \in S_e$,

$$\pi_{\mathbf{S^T}}(\alpha(m_e)(x) > \pi_{\mathbf{S^T}}(x),$$

where $\pi_{\mathbf{S^T}}$ denotes the projection onto the control states space.

A *computational $\alpha$-treeing* is an $\alpha$-treeing $T$ which is a computational $\alpha$-graphing with the distinguished states $\top$, $\bot$ being incomparable maximal elements of the state space.

### *3.2. Quantitative Soundness*

As mentioned in the introduction, we will use in this paper the property of *quantitative soundness* of the dynamic semantics just introduced. This result is essential, as it connects the time complexity of programs in the model considered (e.g. PRAMs, algebraic computation trees) with the length of the orbits of the considered dynamical system. We here only state quantitative soundness for *computational graphings*, i.e. graphings that have distinguished states $\top$ and $\bot$

[6] When $\alpha$ is a group action acting by measure-preserving transformations, this is a *Borel equivalence relation* $\mathcal{R}$, and the condition stated here boils down to requiring that $f$ belongs to the *full group* of $\alpha$.

representing acceptance and rejection respectively. In other words, we consider graphings which compute *decision problems*.

Quantitative soundness is expressed with respect to a translation of machines as graphings, together with a translation of inputs as points of the configuration space. In the following section, these operations are defined for each model of computation considered in this paper. In all these cases, the representation of inputs is straightforward.

**Definition 10.** Let AMC $\alpha$ be an abstract model of computation, and $\mathbb{M}$ a model of computation. A *translation* of $\mathbb{M}$ w.r.t. $\alpha$ is a pair of maps $[\![\cdot]\!]$ which associate to each machine $M$ in $\mathbb{M}$ computing a decision problem a computational $\alpha$-graphing $[\![M]\!]$ and to each input $\iota$ a point $[\![\iota]\!]$ in $\mathbf{X} \times \mathbf{S}$.

**Definition 11.** Let AMC $\alpha$ be an abstract model of computation, $\mathbb{M}$ a model of computation. The AMC $\alpha$ is *quantitatively sound* for $\mathbb{M}$ w.r.t. a translation $[\![\cdot]\!]$ if for all machine $M$ computing a decision problem and input $\iota$, $M$ accepts $\iota$ (resp. rejects $\iota$) in $k$ steps if and only if $[\![M]\!]^k([\![\iota]\!]) = \top$ (resp. $[\![M]\!]^k([\![\iota]\!]) = \bot$).

### *3.3. The algebraic AMCs*

We now define the actions $\alpha_{\text{full}}$ and $\alpha_{\mathbf{R}\text{full}}$. Those will capture all algebraic models of computation considered in this paper, and the main lemma (lemma 5) will be stated for this monoid action. All lower bounds results recovered from the literature, as well as the new lower bounds obtained in this work will be obtained as corollaries of this technical lemma.

As we intend to consider PRAMs at some point, we consider from the beginning the memory of our machines to be separated in two infinite blocks $\mathbf{Z}^\omega$, intended to represent sets of both *shared* and *private* memory cells[7].

**Definition 12.** The underlying space of $\alpha_{\text{full}}$ is $\mathbf{X} = \mathbf{Z}^{\mathbf{Z}} \cong \mathbf{Z}^\omega \times \mathbf{Z}^\omega$. The set of generators is defined by their action on the underlying space, writing $k//n$ the floor $\lfloor k/n \rfloor$ of $k/n$ with the conventions that $k//n = 0$ when $n = 0$ and $\sqrt[n]{k} = 0$ when $k \leqslant 0$:

- $\texttt{const}_i(c)$ initialises the register $i$ with the constant $c \in \mathbf{Z}$: $\alpha_{\text{full}}(\texttt{const}_i(c))(\vec{x}) = (\vec{x}\{x_i := c\})$;
- $\star_i(j,k)$ ($\star \in \{+,-,\times,//\}$) performs the algebraic operation $\star$ on the values in registers $j$ and $k$ and store the result in register $i$: $\alpha_{\text{full}}(\star_i(j,k))(\vec{x}) = (\vec{x}\{x_i := x_j \star x_k\})$;
- $\star_i^c(j)$ ($\star \in \{+,-,\times,//\}$) performs the algebraic operation $\star$ on the value in register $j$ and the constant $c \in \mathbf{Z}$ and store the result in register $i$: $\alpha_{\text{full}}(\star_i^c(j))(\vec{x}) = (\vec{x}\{x_i := c \star x_j\})$;

[7] Obviously, this could be done without any explicit separation of the underlying space, but this will ease the constructions of the next section.

- $\texttt{copy}(i,j)$ copies the value stored in register $j$ in register $i$: $\alpha_{\text{full}}(\texttt{copy}(i,j))(\vec{x}) = (\vec{x}\{x_i := x_j\})$;
- $\texttt{copy}(\sharp i,j)$ copies the value stored in register $j$ in the register whose index is the value stored in register $i$: $\alpha_{\text{full}}(\texttt{copy}(\sharp i,j))(\vec{x}) = (\vec{x}\{x_{x_i} := x_j\})$;
- $\texttt{copy}(i,\sharp j)$ copies the value stored in the register whose index is the value stored in register $j$ in register $i$: $\alpha_{\text{full}}(\texttt{copy}(i,\sharp j))(\vec{x}) = (\vec{x}\{x_i := x_{x_j}\})$;
- $\sqrt[n]{i}(j)$ computes the floor of the $n$-th root of the value stored in register $j$ and store the result in register $i$: $\alpha_{\text{full}}(\sqrt[n]{i}(j))(\vec{x}) = (\vec{x}\{x_i := \sqrt[n]{x_j}\})$.

We also define the real-valued equivalent, which will be essential for the proof of lower bounds. The corresponding AMC $\alpha_{\mathbf{R}\text{ram}}$ is defined in the same way than the integer-valued one, but with underlying space $\mathbf{X} = \mathbf{R}^{\mathbf{Z}}$ and with instructions adapted accordingly:

- the division and $n$-th root operations are the usual operations on the reals;
- the three copy operators are only effective on integers.

Note that we consider the space $\mathbf{R}^{\mathbf{R}}$, i.e. an uncountable number of potential registers. This appears to us as the simplest way to represent the model of real-valued PRAMs which includes indirect addressing. In practise, only a finite number of registers can be accessed during a finite execution (since indexes need to be computed), and therefore a countable number of potential registers would be enough. However this poses the issue of defining the semantics properly: using maps from $\mathbf{R}$ to $\mathbf{N}$ do not work because this creates side-effects giving more expressive power to the machines (e.g. considering that indirect addressing $\texttt{copy}(\sharp i,j)$ modifies the register of index $\lfloor i \rfloor$ – where $\lfloor \cdot \rfloor$ is the floor function – turns out to provide a way to define euclidean division!). On the other hand, defining a dynamic allocation of register should be possible but would complicate the definitions.

**Definition 13.** The underlying space of $\alpha_{\mathbf{R}\text{full}}$ is $\mathbf{X} = \mathbf{R}^{\mathbf{R}} \cong \mathbf{R}^{\mathbf{R}} \times \mathbf{R}^{\mathbf{R}}$. The set of generators is defined by their action on the underlying space, with the conventions that $k/n = 0$ when $n = 0$ and $\sqrt[n]{k} = 0$ when $k \leqslant 0$:

- $\texttt{const}_i(c)$ initialises the register $i$ with the constant $c \in \mathbf{R}$: $\alpha_{\mathbf{R}\text{full}}(\texttt{const}_i(c))(\vec{x}) = (\vec{x}\{x_i := c\})$;
- $\star_i(j,k)$ ($\star \in \{+,-,\times,/\}$) performs the algebraic operation $\star$ on the values in registers $j$ and $k$ and store the result in register $i$: $\alpha_{\mathbf{R}\text{full}}(\star_i(j,k))(\vec{x}) = (\vec{x}\{x_i := x_j \star x_k\})$;
- $\star_i^c(j)$ ($\star \in \{+,-,\times,/\}$) performs the algebraic operation $\star$ on the value in register $j$ and the constant $c \in \mathbf{R}$ and store the result in register $i$: $\alpha_{\mathbf{R}\text{full}}(\star_i^c(j))(\vec{x}) = (\vec{x}\{x_i := c \star x_j\})$;
- $\texttt{copy}(i,j)$ copies the value stored in register $j$ in register $i$: $\alpha_{\mathbf{R}\text{full}}(\texttt{copy}(i,j))(\vec{x}) = (\vec{x}\{x_i := x_j\})$;

- $\texttt{copy}(\sharp i, j)$ copies the value stored in register $j$ in the register whose index is the value stored in register $i$: $\alpha_{\mathbf{R}\text{full}}(\texttt{copy}(\sharp i, j))(\vec{x}) = (\vec{x}\{x_{x_i} := x_j\})$;
- $\texttt{copy}(i, \sharp j)$ copies the value stored in register $j$ in register whose index is the value stored in register $i$: $\alpha_{\mathbf{R}\text{full}}(\texttt{copy}(i, \sharp j))(\vec{x}) = (\vec{x}\{x_i := x_{x_j}\})$;
- $\sqrt[n]{i}(j)$ computes the $n$-th real root of the value stored in register $j$ and store the result in register $i$: $\alpha_{\mathbf{R}\text{full}}(\sqrt[n]{i}(j))(\vec{x}) = (\vec{x}\{x_i := \sqrt[n]{x_j}\})$.

## 4. Algebraic models of computations as AMCs

### *4.1. Algebraic computation trees*

The first model considered here will be that of *algebraic computation tree* as defined by Ben-Or [2]. Let us note this model refines the *algebraic decision trees* model of Steele and Yao [1], a model of computation consisting in binary trees for which each branching performs a test w.r.t. a polynomial and each leaf is labelled `YES` or `NO`. Algebraic computation trees only allow tests w.r.t. 0, while additional vertices corresponding to algebraic operations can be used to construct polynomials.

**Definition 14** (algebraic computation trees, [2])**.** An *algebraic computation tree* on $\mathbf{R}^n$ is a binary tree $T$ with an function assigning:

- to any vertex $v$ with only one child (simple vertex) an operational instruction of the form $f_v = f_{v_i} \star f_{v_j}$, $f_v = c \star f_{v_i}$, or $f_v = \sqrt{f_{v_i}}$, where $\star \in \{+, -, \times, /\}$, $v_i, v_j$ are ancestors of $v$ and $c \in \mathbf{R}$ is a constant;
- to any vertex $v$ with two children a test instruction of the form $f_{v_i} \star 0$, where $\star \in \{>, =, \geqslant\}$, and $v_i$ is an ancestor of $v$ or $f_{v_i} \in \{x_1, \ldots, x_n\}$;
- to any leaf an output `YES` or `NO`.

Let $W \subseteq \mathbf{R}^n$ be any set and $T$ be an algebraic computation tree. We say that $T$ computes the membership problem for $W$ if for all $x \in \mathbf{R}^n$, the traversal of $T$ following $x$ ends on a leaf labelled `YES` if and only if $x \in W$.

As algebraic computation trees are *trees*, they will be represented by treeings, i.e. $\alpha_{\mathbf{R}\text{full}}$-programs whose set of control states can be ordered so that any edge in the graphing is strictly increasing on its control states component.

**Definition 15.** Let $T$ be a computational $\alpha_{\mathbf{R}\text{full}}$-treeing. The set of inputs $\mathrm{In}(T)$ (resp. outputs $\mathrm{Out}(T)$) is the set of integers $k$ (resp. $i$) such that there exists an edge $e$ in $T$ satisfying that:

- either $e$ is realised by one of $+_i(j,k)$, $+_i(k,j)$, $-_i(j,k)$, $-_i(k,j)$, $\times_i(j,k)$, $\times_i(k,j)$, $/_i(j,k)$, $/_i(k,j)$ $+^c_i(k)$, $-^c_i(k)$, $\times^c_i(k)$, $/^c_i(k)$, $\sqrt[n]{i}(k)$;
- or the source of $e$ is one among $\mathbf{R}^\omega_{k\geqslant 0}$, $\mathbf{R}^\omega_{k\leqslant 0}$, $\mathbf{R}^\omega_{k>0}$, $\mathbf{R}^\omega_{k<0}$, $\mathbf{R}^\omega_{k=0}$, and $\mathbf{R}^\omega_{k\neq 0}$.

The *effective input space* $\mathbf{In}^{\mathrm{E}}(T)$ of an $\alpha_{\mathrm{act}}$-treeing $T$ is defined as the set of indices $k \in \omega$ belonging to $\mathrm{In}(T)$ but not to $\mathrm{Out}(T)$. The *implicit input space* $\mathbf{In}^{\mathrm{I}}(T)$ of an $\alpha_{\mathrm{act}}$-treeing $T$ is defined as the set of indices $k \in \omega$ such that $k \notin \mathrm{Out}(T)$.

**Definition 16.** Let $T$ be an $\alpha_{\mathbf{R}\mathrm{full}}$-treeing, and assume that $1, 2, \ldots, n \in \mathbf{In}^{\mathrm{I}}(T)$. We say that $T$ computes the membership problem for $W \subseteq \mathbf{R}^n$ in $k$ steps if $k$ successive iterations of $T$ restricted to $\{(x_i)_{i\in\omega} \in \mathbf{R}^\omega \mid \forall 1 \leqslant i \leqslant n, x_i = y_i\} \times \{0\}$ reach state $\top$ if and only if $(y_1, y_2, \ldots, y_n) \in W$.

*Remark.* Let $\vec{x} = (x_1, x_2, \ldots, x_n)$ be an element of $\mathbf{R}^n$ and consider two elements $a, b$ in the subspace $\{(y_1, \ldots, y_n, \ldots) \in \mathbf{R}^\omega \mid \forall 1 \geqslant i \geqslant n, y_i = x_i\} \times \{0\}$. One easily checks that $\pi_{\mathbf{S}}(T^k(a)) = \top$ if and only if $\pi_{\mathbf{S}}(T^k(b)) = \top$, where $\pi_{\mathbf{S}}$ is the projection onto the state space and $T^k(a)$ represents the $k$-th iteration of $T$ on $a$. It is therefore possible to consider only a standard representative $[\![\vec{x}]\!]$ of $\vec{x} \in \mathbf{R}^n$, for instance $(x_1, \ldots, x_n, 0, 0, \ldots) \in \mathbf{R}^\omega$, to decide whether $\vec{x}$ is accepted by $T$.

**Definition 17.** Let $T$ be an algebraic computation tree on $\mathbf{R}^n$, and $T^\circ$ be the associated directed acyclic graph, built from $T$ by merging all the leaves tagged `YES` in one leaf $\top$ and all the leaves tagged `NO` in one leaf $\bot$. Suppose the internal vertices are numbered $\{n+1, \ldots, n+\ell\}$; the numbers $1, \ldots, n$ being reserved for the input.

We define $[\![T]\!]$ as the $\alpha_{\mathrm{act}}$-graphing with control states $\{n+1, \ldots, n+\ell, \top, \bot\}$ and where each internal vertex $i$ of $T^\circ$ defines either:

- a single edge of source $\mathbf{R}^\omega$ realized by:
  - $(\star_i(j,k), i \mapsto t)$ $(\star \in \{+,-,\times\})$ if $i$ is associated to $f_{v_i} = f_{v_j} \star f_{v_k}$ and $t$ is the child of $i$;
  - $(\star_i^c(j), i \mapsto t)$ $(\star \in \{+,-,\times\})$ if $i$ is associated to $f_{v_i} = c \star f_{v_k}$ and $t$ is the child of $i$;
- a single edge of source $\mathbf{R}^\omega_{k\neq 0}$ realized by:
  - $(/_i(j,k), i \mapsto t)$ if $i$ is associated to $f_{v_i} = f_{v_j}/f_{v_k}$ and $t$ is the child of $i$;
  - $(/_i^c(k), i \mapsto t)$ if $i$ is associated to $f_{v_i} = c/f_{v_k}$ and $t$ is the child of $i$;
- a single edge of source $\mathbf{R}^\omega_{k\geqslant 0} \times \{i\}$ realized by $(\sqrt[2]{i}(k), i \mapsto t)$ if $i$ is associated to $f_{v_i} = \sqrt{f_{v_k}}$ and $t$ is the child of $i$;
- two edges if $i$ is associated to $f_{v_i} \star 0$ (where $\star$ ranges in $>$, $\geqslant$) and its two sons are $j$ and $k$. Those are of respective sources $\mathbf{R}^\omega_{k\star 0} \times \{i\}$ and $\mathbf{R}^\omega_{k\bar{\star}0} \times \{i\}$ (where $\bar{\star} =' \leqslant'$ if $\star =' >'$, $\bar{\star} =' <'$ if $\star =' \geqslant'$, and $\bar{\star} =' \neq'$ if $\star =' =' .$), respectively realized by $(\mathrm{Id}, i \mapsto j)$ and $(\mathrm{Id}, i \mapsto k)$

**Proposition 18.** *Any algebraic computation tree $T$ of depth $k$ is faithfully and quantitatively interpreted as the $\alpha_{\mathbf{R}\mathrm{full}}$-program $[\![T]\!]$. I.e. $T$ computes the membership problem for $W \subseteq \mathbf{R}^n$ if and only if $[\![T]\!]$ computes the membership problem for $W$ in $k$ steps – that is $\pi_{\mathbf{S}}([\![T]\!]^k([\![\vec{x}]\!])) = \top$.*

As a corollary of this proposition, we get quantitative soundness.

**Theorem 19.** *The representation of* ACTs *as $\alpha_{\mathbf{R}\text{full}}$-programs is quantitatively sound.*

*4.2. Algebraic circuits*

As we will recover Cucker's proof that $\mathrm{NC}_{\mathbf{R}} \neq \mathrm{PTIME}_{\mathbf{R}}$, we introduce the model of *algebraic circuits* and their representation as $\alpha_{\mathbf{R}\text{full}}$-programs.

**Definition 20.** An algebraic circuit over the reals with inputs in $\mathbf{R}^n$ is a finite directed graph whose vertices have labels in $\mathbf{N} \times \mathbf{N}$, that satisfies the following conditions:

- There are exactly $n$ vertices $v_{0,1}, v_{0,2}, \ldots, v_{0,n}$ with first index 0, and they have no incoming edges;
- all the other vertices $v_{i,j}$ are of one of the following types:
  1. arithmetic vertex: they have an associated arithmetic operation $\{+, -, \times, /\}$ and there exist natural numbers $l, k, r, m$ with $l, k < i$ such that their two incoming edges are of sources $v_{l,r}$ and $v_{k,m}$;
  2. constant vertex: they have an associated real number $y$ and no incoming edges;
  3. sign vertex: they have a unique incoming edge of source $v_{k,m}$ with $k < i$.

We call *depth* of the circuit the largest $m$ such that there exist a vertex $v_{m,r}$, and *size* of the circuit the total number of vertices. A circuit of depth $d$ is *decisional* if there is only one vertex $v_{d,r}$ at level $d$, and it is a sign vertex; we call $v_{d,r}$ the *end vertex* of the decisional circuit.

To each vertex $v$ one inductively associates a function $f_v$ of the input variables in the usual way, where a sign node with input $x$ returns 1 if $x > 0$ and 0 otherwise. The accepted set of a decisional circuit $C$ is defined as the set $S \subseteq \mathbf{R}^n$ of points whose image by the associated function is 1, i.e. $S = f_v^{-1}(\{1\})$ where $v$ is the end vertex of $C$.

We represent algebraic circuit as computational $\alpha_{\mathbf{R}\text{full}}$-treeings as follows. The first index in the pairs $(i, j) \in \mathbf{N} \times \mathbf{N}$ are represented as states, the second index is represented as an index in the infinite product $\mathbf{R}^\omega$, and vertices are represented as edges.

**Definition 21.** Let $C$ be an algebraic circuit, defined as a finite directed graph $(V, E, s, t, \ell)$ where $V \subset \mathbf{N} \times \mathbf{N}$, and $\ell : V \to \{\mathrm{init}, +, -, \times, /, \mathrm{sgn}\} \cup \{\mathrm{const}_c \mid c \in \mathbf{R}\}$ is a vertex labelling map. We suppose without loss of generality that for each $j \in \mathbf{N}$, there is at most one $i \in \mathbf{N}$ such that $(i, j) \in V$. We define $N$ as $\max\{j \in \mathbf{N} \mid \exists i \in \mathbf{N}, (i, j) \in V\}$.

We define the $\alpha_{\mathbf{R}\text{full}}$-program $[\![C]\!]$ by choosing as set of control states $\{i \in \mathbf{N} \mid \exists j \in \mathbf{N}, (i, j) \in V\}$ and the collection of edges $\{e_{(i,j)} \mid i \in \mathbf{N}^*, j \in \mathbf{N}, (i, j) \in V\} \cup \{e^+_{(i,j)} \mid i \in \mathbf{N}^*, j \in \mathbf{N}, (i, j) \in V, \ell(v) = \mathrm{sgn}\}$ realised as follows:

- if $\ell(v) = \mathrm{const}_c$, the edge $e_{(i,j)}$ is realised as $(+_j^{n_v}(c), 0 \mapsto i)$ of source $\mathbf{R}^\omega_{n_v=0} \times \{0\}$;
- if $\ell(v) = \star\, (\star \in \{+, -, \times\})$ of incoming edges $(k, l)$ and $(k', l')$, the edge $e_{(i,j)}$ is of source $\mathbf{R}^\omega \times \{\max(k, k')\}$ and realised by $(\star_j(l, l'), \max(k, k') \mapsto i)$;
- if $\ell(v) = /$ of incoming edges $(k, l)$ and $(k', l')$, the edge $e_{(i,j)}$ is of source $\mathbf{R}^\omega_{l' \neq 0} \times \{\max(k, k')\}$ and realised by $(/_j(l, l'), \max(k, k') \mapsto i)$;
- if $\ell(v) = \mathrm{sgn}$ of incoming edge $(k, l)$, the edges $e_{(i,j)}$ and $e^+_{(i,j)}$ are of respective sources $\mathbf{R}^\omega_{n_v=0 \wedge x_l \leqslant 0} \times \{k\}$ and $\mathbf{R}^\omega_{n_v=0 \wedge x_l > 0} \times \{k\}$ realised by $(\mathrm{Id}, k \mapsto i)$ and $(+_j(n_v, 1), k \mapsto i)$ respectively.

As each step of computation in the algebraic circuit is translated as going through a single edge in the corresponding $\alpha_{\mathbf{R}\mathrm{full}}$-program, the following result is straightforward.

**Theorem 22.** *The representation of* ALGCIRC *as $\alpha_{\mathbf{R}\mathrm{full}}$-programs is quantitatively sound.*

*4.3. Algebraic* RAMs

In this paper, we will consider algebraic parallel random access machines, that act not on strings of bits, but on integers. In order to define those properly, we first define the notion of (sequential) random access machine (RAM) before considering their parallelisation.

A RAM *command* is a pair $(\ell, I)$ of a *line* $\ell \in \mathbf{N}^\star$ and an *instruction* $\mathtt{I}$ among the following, where $i, j \in \mathbf{N}$, $\star \in \{+, -, \times, /\}$, $c \in \mathbf{Z}$ is a constant and $\ell, \ell' \in \mathbf{N}^\star$ are lines:

$$\mathtt{skip};\quad \mathtt{X_i} \coloneqq \mathtt{c};\quad \mathtt{X_i} \coloneqq \mathtt{X_j} \star \mathtt{X_k};\quad \mathtt{X_i} \coloneqq \mathtt{X_j};$$
$$\mathtt{X_i} \coloneqq \sharp\mathtt{X_j};\quad \sharp\mathtt{X_i} \coloneqq \mathtt{X_j};\quad \mathtt{if\ X_i = 0\ goto}\ \ell\ \mathtt{else}\ \ell'.$$

A RAM *machine* $M$ is then a finite set of commands such that the set of lines is $\{1, 2, \ldots, |M|\}$, with $|M|$ the *length* of $M$. We will denote the commands in $M$ by $(i, \mathrm{Inst}_M(i))$, i.e. $\mathrm{Inst}_M(i)$ denotes the line $i$ instruction.

Following Mulmuley [4], we will here make the assumption that the input in the RAM (and in the PRAM model defined in the next section) is split into *numeric* and *nonumeric* data – e.g. in the `maxflow` problem the nonnumeric data would specify the network and the numeric data would specify the edge-capacities – and that indirect references use pointers depending only on nonnumeric data[8]. We refer the reader to Mulmuley's article for more details.

[8] Quoting Mulmuley: "We assume that the pointer involved in an indirect reference is not some numeric argument in the input or a quantity that depends on it. For example, in the max- flow problem the algorithm should not use an edge-capacity as a pointer—which is a reasonable condition. To enforce this restriction, one initially puts an invalid-pointer tag on every numeric argument in the input. During the execution of an arithmetic instruction, the same tag is also propagated to the result if any operand has that tag. Trying to use a memory value with invalid-pointer tag results in error." [4, Page 1468].

Machines in the RAM model can be represented as graphings w.r.t. the action $\alpha_{\text{full}}$. Intuitively the encoding works as follows. The notion of *control state* allows to represent the notion of *line* in the program. Then, the action just defined allows for the representation of all commands but the conditionals. The conditionals are represented as follows: depending on the value of $X_i$ one wants to jumps either to the line $\ell$ or to the line $\ell'$; this is easily modelled by two different edges of respective sources $\mathbb{H}(i) = \{\vec{x} \mid x_i = 0\}$ and $\mathbb{H}(i)^{\mathrm{c}} = \{\vec{x} \mid x_i \neq 0\}$.

**Definition 23.** Let $M$ be a RAM machine. We define the translation $[\![M]\!]$ as the $\alpha_{\text{ram}}$-program with set of control states $\{0, 1, \dots, L, L+1\}$ where each line $\ell$ defines (in the following, $\star \in \{+, -, \times\}$ and we write $\ell_{++}$ the map $\ell \mapsto \ell + 1$):

- a single edge $e$ of source $\mathbf{X} \times \{\ell\}$ and realised by:
  - $(\mathrm{Id}, \ell_{++})$ if $\mathrm{Inst}_M(\ell) = \texttt{skip}$;
  - $(\texttt{const}_i(c), \ell_{++})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \mathtt{c}$;
  - $(\star_i(j,k), \ell_{++})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \mathtt{X_j} \star \mathtt{X_k}$;
  - $(\texttt{copy}(i,j), \ell_{++})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \mathtt{X_j}$;
  - $(\texttt{copy}(i, \sharp j), \ell_{++})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \sharp\mathtt{X_j}$;
  - $(\texttt{copy}(\sharp i, j), \ell_{++})$ if $\mathrm{Inst}_M(\ell) = \sharp\mathtt{X_i} \coloneqq \mathtt{X_j}$.
- an edge $e$ of source $\mathbb{H}(k)^{\mathrm{c}} \times \{\ell\}$ realised by $(//_i(j,k), \ell_{++})$ if $\mathrm{Inst}_M(\ell)$ is $\mathtt{X_i} \coloneqq \mathtt{X_j}/\mathtt{X_k}$;
- a pair of edges $e, e^{\mathrm{c}}$ of respective sources $\mathbb{H}(i) \times \{\ell\}$ and $\mathbb{H}(i)^{\mathrm{c}} \times \{\ell\}$ and realised by respectively $(\mathrm{Id}, \ell \mapsto \ell^0)$ and $(\mathrm{Id}, \ell \mapsto \ell^1)$, if the line is a conditional $\texttt{if } \mathtt{X_i} = \mathtt{0} \texttt{ goto } \ell^0 \texttt{ else } \ell^1$.

The translation $[\![\iota]\!]$ of an input $\iota \in \mathbf{Z}^d$ is the point $(\bar{\iota}, 0)$ where $\bar{\iota}$ is the sequence $(\iota_1, \iota_2, \dots, \iota_k, 0, 0, \dots)$.

Now, the main result for the representation of RAMs is the following. The proof is straightforward, as each instruction corresponds to exactly one edge, except for the conditional case (but given a configuration, it lies in the source of at most one of the two edges translating the conditional).

**Theorem 24.** *The representation of* RAMs *as* $\alpha_{\text{full}}$*-programs is quantitatively sound w.r.t. the translation just defined.*

### *4.4. The Crew operation and* PRAMs

Based on the notion of RAM, we are now able to consider their parallelisation, namely PRAMs. A PRAM $M$ is given as a finite sequence of RAM machines $M_1, \dots, M_p$, where $p$ is the number of *processors* of $M$. Each processor $M_i$ has access to its own, private, set of registers $(\mathtt{X}^i_k)_{k \geqslant 0}$ and a *shared memory* represented as a set of registers $(\mathtt{X}^0_k)_{k \geqslant 0}$.

One has to deal with conflicts when several processors try to access the shared memory simultaneously. We here chose to work with the *Concurrent Read, Exclusive Write* (CREW) discipline: at a given step at which several processors try to write in the shared memory, only the processor with the smallest index will be allowed to do so. In order to model such parallel computations, we abstract the CREW at the level of monoids. For this, we suppose that we have two monoid actions $\mathrm{M}\langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X} \times \mathbf{Y}$ and $\mathrm{M}\langle H, \mathrm{Q}\rangle \curvearrowright \mathbf{X} \times \mathbf{Z}$, where $\mathbf{X}$ represents the

shared memory. We then consider the subset $\# \subset G \times H$ of pairs of generators that potentially conflict with one another – the conflict relation.

**Definition 25** (Conflicted sum)**.** Let $\mathrm{M}\langle G, \mathrm{R}\rangle$, $\mathrm{M}\langle G', \mathrm{R}'\rangle$ be two monoids and $\# \subseteq G \times G'$. The *conflicted sum of* $\mathrm{M}\langle G, \mathrm{R}\rangle$ *and* $\mathrm{M}\langle G', \mathrm{R}'\rangle$ *over* $\#$, noted $\mathrm{M}\langle G, \mathrm{R}\rangle *_{\#} \mathrm{M}\langle G', \mathrm{R}'\rangle$, is defined as the monoid with generators $(\{1\} \times G) \cup (\{2\} \times G')$ and relations

$$\begin{array}{c}(\{1\} \times R) \cup (\{2\} \times R') \cup \{(\mathbf{1}, e)\} \cup \{(\mathbf{1}, e')\} \\ \cup \{\big((1,g)(2,g'), (2,g')(1,g)\big) \mid (g,g') \notin \#\}\end{array}$$

where $\mathbf{1}$, $e$, $e'$ are the units of $\mathrm{M}\langle G, \mathrm{R}\rangle *_{\#} \mathrm{M}\langle G', \mathrm{R}'\rangle$, $\mathrm{M}\langle G, \mathrm{R}\rangle$ and $\mathrm{M}\langle G', \mathrm{R}'\rangle$ respectively.

In the particular case where $\# = (G \times H') \cup (H \times G')$, with $H, H'$ respectively subsets of $G$ and $G'$, we will write the sum $\mathrm{M}\langle G, \mathrm{R}\rangle \ {}_H{*}_{H'} \mathrm{M}\langle G', \mathrm{R}'\rangle$.

*Remark.* When the conflict relation $\#$ is empty, this defines the usual direct product of monoids. This corresponds to the case in which no conflicts can arise w.r.t. the shared memory. In other words, the direct product of monoids corresponds to the parallelisation of processes *without shared memory*.

Dually, when the conflict relation is full ($\# = G \times G'$), this defines the free product of the monoids.

**Definition 26.** Let $\alpha : M \curvearrowright \mathbf{X} \times \mathbf{Y}$ be a monoid action. We say that an element $m \in M$ is *central relatively to* $\alpha$ (or just *central*) if $m$ acts as the identity on $\mathbf{X}$, i.e.[9] $\alpha(m); \pi_X = \pi_X$.

Intuitively, central elements are those that will not affect the shared memory. As such, only *non-central elements* require care when putting processes in parallel.

**Definition 27.** Let $\mathrm{M}\langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X} \times \mathbf{Y}$ be an AMC. We note $Z_\alpha$ the set of central elements and $\bar{Z}_\alpha(G) = \{m \in G \mid m \notin Z_\alpha\}$.

**Definition 28** (The CREW of AMCs)**.** Let $\alpha : \mathrm{M}\langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X} \times \mathbf{Y}$ and $\beta : \mathrm{M}\langle H, \mathrm{Q}\rangle \curvearrowright \mathbf{X} \times \mathbf{Z}$ be AMCs. We define the AMC $\mathrm{CREW}(\alpha, \beta) : \mathrm{M}\langle G, \mathrm{R}\rangle \ {}_{\bar{Z}_\alpha(G)}{*}_{\bar{Z}_\beta(G')} \mathrm{M}\langle G', \mathrm{R}'\rangle \curvearrowright \mathbf{X} \times \mathbf{Y} \times \mathbf{Z}$ by letting $\mathrm{CREW}(\alpha, \beta)(m, m') = \alpha(m) * \beta(m')$ on elements of $G \times G'$, where[10]:

$$\alpha(m) * \beta(m') = \begin{cases} \Delta_1; [\alpha(m); \pi_Y, \beta(m')]; [\sigma_{X,Y}, \mathrm{Id}Z] & \text{if } m \notin \bar{Z}_\alpha(G), m' \in \bar{Z}_\beta(G'), \\ \Delta_2; [\alpha(m), \beta(m'); \pi_Z] & \text{otherwise,} \end{cases}$$

[9] Here and in the following, we denote by ; the sequential composition of functions. I.e. $f; g$ denotes what is usually written $g \circ f$.

[10] We denote $\sigma_{X,Y} : \mathbf{Y} \times \mathbf{X} \to \mathbf{X} \times \mathbf{Y}$ the map defined as $(y, x) \mapsto (x, y)$.

with $\Delta_i : \mathbf{X} \times \mathbf{Y} \times \mathbf{Z} \to \mathbf{X} \times \mathbf{Y} \times \mathbf{X} \times \mathbf{Z}$ defined[11] as:

$$\Delta_1 : (x, y, z) \mapsto (x_0, y, x, z)$$
$$\Delta_2 : (x, y, z) \mapsto (x, y, x_0, z).$$

We can now define AMC of PRAMs and thus the interpretations of PRAMs as abstract programs. For each integer $p$, we define the AMC $\text{CREW}^p(\alpha_{\text{full}})$. This allows the consideration of up to $p$ parallel RAMs: the translation of such a RAM with $p$ processors is defined by extending the translation of RAMs by considering a set of states equal to $L_1 \times L_2 \times \cdots \times L_p$ where for all $i$ the set $L_i$ is the set of lines of the $i$-th processor.

Now, to deal with arbitrary large PRAMs, i.e. with arbitrarily large number of processors, one considers the following AMC defined as a *direct limit*.

**Definition 29** (The AMC of PRAMs)**.** Let $\alpha : M \curvearrowright \mathbf{X} \times \mathbf{X}$ be the AMC $\alpha_{\text{full}}$. The AMC of PRAMs is defined as $\alpha_{\text{pram}} = \varinjlim \text{CREW}^k(\alpha)$, where $\text{CREW}^{k-1}(\alpha)$ is identified with a restriction of $\text{CREW}^k(\alpha)$ through $\text{CREW}^{k-1}(\alpha)(m_1, \ldots, m_{k-1}) \mapsto \text{CREW}^k(\alpha)(m_1, \ldots, m_{k-1}, 1)$.

Remark that the underlying space of the PRAM AMC $\alpha_{\text{pram}}$ is defined as the union $\cup_{n \in \omega} \mathbf{Z}^\omega \times (\mathbf{Z}^\omega)^n$ which we will write $\mathbf{Z}^\omega \times (\mathbf{Z}^\omega)^{(\omega)}$. In practise a given $\alpha_{\text{pram}}$-program admitting a finite $\alpha_{\text{pram}}$ representative will only use elements in $\text{CREW}^p(\alpha_{\text{full}})$, and can therefore be understood as a $\text{CREW}^p(\alpha)$-program.

**Theorem 30.** *The representation of* PRAMs *as $\alpha_{\text{pram}}$-programs is quantitatively sound.*

### *4.5. Real* PRAMs

These definitions and results stated for integer-valued PRAMs can be adapted to define *real-valued* PRAMs and their translation as $\alpha_{\mathbf{R}\text{full}}$-programs.

A real-valued RAM *command* is a pair $(\ell, I)$ of a *line* $\ell \in \mathbf{N}^\star$ and an *instruction* $\mathtt{I}$ among the following, where $i, j \in \mathbf{N}$, $\star \in \{+, -, \times, /\}$, $c \in \mathbf{Z}$ is a constant and $\ell, \ell' \in \mathbf{N}^\star$ are lines:

$$\mathtt{skip}; \quad \mathtt{X_i} \coloneqq \mathtt{c}; \quad \mathtt{X_i} \coloneqq \mathtt{X_j} \star \mathtt{X_k}; \quad \mathtt{X_i} \coloneqq \mathtt{X_j};$$
$$\mathtt{X_i} \coloneqq \sharp\mathtt{X_j}; \quad \sharp\mathtt{X_i} \coloneqq \mathtt{X_j}; \quad \mathtt{if\ X_i = 0\ goto}\ \ell\ \mathtt{else}\ \ell'.$$

We consider a restriction for pointers similar to that considered in the case of integer-valued RAMs. A real-valued RAM *machine* $M$ is then a finite set of commands such that the set of lines is $\{1, 2, \ldots, |M|\}$, with $|M|$ the *length* of $M$. We will denote the commands in $M$ by $(i, \text{Inst}_M(i))$, i.e. $\text{Inst}_M(i)$ denotes the line $i$ instruction.

A real-valued PRAM $M$ is given as a finite sequence of real-valued RAM machines $M_1, \ldots, M_p$, where $p$ is the number of *processors* of $M$. Each processor

[11] Formally, the definition of $\Delta_i$ is parametrised by the choice of a point $x_0 \in \mathbf{X}$, but the map $\alpha(m) * \beta(m')$ does not depend on this choice because of the projections on $\mathbf{Y}$ and $\mathbf{Z}$.

$M_i$ has access to its own, private, set of registers $(\mathtt{X}^i_k)_{k\geqslant 0}$ and a *shared memory* represented as a set of registers $(\mathtt{X}^0_k)_{k\geqslant 0}$. Again, we chose to work with the *Concurrent Read, Exclusive Write* (CREW) discipline as it is well translated through the CREW operation of AMCs.

The following definition provides the straightforward definition of real-valued PRAMs as graphings.

**Definition 31.** Let $M$ be a real-valued RAM machine. We define the translation $[\![M]\!]$ as the $\alpha_{\mathbf{R}\text{full}}$-program with set of control states $\{0, 1, \dots, L, L+1\}$ where each line $\ell$ defines (in the following, $\star \in \{+, -, \times\}$ and we write $\ell{+}{+}$ the map $\ell \mapsto \ell + 1$):

- a single edge $e$ of source $\mathbf{X} \times \{\ell\}$ and realised by:
  - $(\mathrm{Id}, \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \mathtt{skip}$;
  - $(\mathtt{const}_i(c), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \mathtt{c}$;
  - $(\star_i(j,k), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \mathtt{X_j} \star \mathtt{X_k}$;
  - $(\mathtt{copy}(i,j), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \mathtt{X_j}$;
  - $(\mathtt{copy}(i, \sharp j), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \sharp\mathtt{X_j}$;
  - $(\mathtt{copy}(\sharp i, j), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \sharp\mathtt{X_i} \coloneqq \mathtt{X_j}$.
- an edge $e$ of source $\mathbf{R}_{k\neq 0} \times \{\ell\}$ realised by $(/_i(j,k), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell)$ is $\mathtt{X_i} \coloneqq \mathtt{X_j}/\mathtt{X_k}$;
- a pair of edges $e, e^{\mathrm{c}}$ of respective sources $\mathbf{R}_{i=0} \times \{\ell\}$ and $\mathbf{R}_{i\neq 0} \times \{\ell\}$ and realised by respectively $(\mathrm{Id}, \ell \mapsto \ell^0)$ and $(\mathrm{Id}, \ell \mapsto \ell^1)$, if the line is a conditional $\mathtt{if\ X_i = 0\ goto}\ \ell^0\ \mathtt{else}\ \ell^1$.

The translation $[\![\iota]\!]$ of an input $\iota \in \mathbf{Z}^d$ is the point $(\bar{\iota}, 0)$ where $\bar{\iota}$ is the sequence $(\iota_1, \iota_2, \dots, \iota_k, 0, 0, \dots)$.

For each integer $p$, we then define the AMC $\mathrm{CREW}^p(\alpha_{\mathbf{R}\text{full}})$. This allows the consideration of up to $p$ parallel real-valued RAMs: the translation of such a RAM with $p$ processors is defined by extending the translation of real-valued RAMs just defined by considering a set of states equal to $L_1 \times L_2 \times \cdots \times L_p$ where for all $i$ the set $L_i$ is the set of lines of the $i$-th processor.

Since we need to translate arbitrary large real-valued PRAMs, i.e. with arbitrarily large number of processors, one considers the following AMC defined as a *direct limit*.

**Definition 32** (The AMC of real-valued PRAMs)**.** Let $\alpha : M \curvearrowright \mathbf{X} \times \mathbf{X}$ be the AMC $\alpha_{\mathbf{R}\text{full}}$. The AMC of real-valued PRAMs is defined as $\alpha_{\mathbf{R}\text{pram}} = \varinjlim \mathrm{CREW}^k(\alpha)$, where $\mathrm{CREW}^{k-1}(\alpha)$ is identified with a restriction of $\mathrm{CREW}^k(\alpha)$ through

$$\mathrm{CREW}^{k-1}(\alpha)(m_1, \dots, m_{k-1}) \mapsto \mathrm{CREW}^k(\alpha)(m_1, \dots, m_{k-1}, 1).$$

Then the following results are quite straightforward.

**Theorem 33.** *The representation of real-valued* RAMs *as $\alpha_{\mathbf{R}\text{full}}$-programs is quantitatively sound. The representation of real-valued* PRAMs *as $\alpha_{\mathbf{R}\text{pram}}$-programs is quantitatively sound.*

## 5. Entropy and Cells

### *5.1. Topological Entropy*

Topological Entropy was introduced in the context of dynamical systems in an attempt to classify the latter w.r.t. conjugacy. The topological entropy of a dynamical system is a value representing the average exponential growth rate of the number of orbit segments distinguishable with a finite (but arbitrarily fine) precision. The definition is based on the notion of open covers.

*Open covers..* Given a topological space $\mathbf{X}$, an *open cover* of $\mathbf{X}$ is a family $\mathcal{U} = (U_i)_{i\in I}$ of open subsets of $\mathbf{X}$ such that $\cup_{i\in I} U_i = \mathbf{X}$. A finite cover $\mathcal{U}$ is a cover whose indexing set is finite. A *subcover* of a cover $\mathcal{U} = (U_i)_{i\in I}$ is a sub-family $\mathcal{S} = (U_j)_{j\in J}$ for $J \subseteq I$ such that $\mathcal{S}$ is a cover, i.e. such that $\cup_{j\in J} U_j = \mathbf{X}$.

We will denote by $\mathrm{Cov}(\mathbf{X})$ (resp. $\mathrm{FCov}(\mathbf{X})$) the set of all open covers (resp. all finite open covers) of the space $\mathbf{X}$.

We now define two operations on open covers that are essential to the definition of entropy. An open cover $\mathcal{U} = (U_i)_{i\in I}$, together with a continuous function $f : \mathbf{X} \to \mathbf{X}$, defines the inverse image open cover $f^{-1}(\mathcal{U}) = (f^{-1}(U_i))_{i\in I}$. Note that if $\mathcal{U}$ is finite, $f^{-1}(\mathcal{U})$ is finite as well. Given two open covers $\mathcal{U} = (U_i)_{i\in I}$ and $\mathcal{V} = (V_j)_{j\in J}$, we define their join $\mathcal{U} \vee \mathcal{V}$ as the family $(U_i \cap V_j)_{(i,j)\in I\times J}$. Once again, if both initial covers are finite, their join is finite.

*Entropy.* Usually, entropy is defined for continuous maps on a compact set, following the original definition by Adler, Konheim and McAndrews [35]. Using the fact that arbitrary open covers have a finite subcover, this allows one to ensure that the smallest subcover of any cover is finite. I.e. given an arbitrary cover $\mathcal{U}$, one can consider the smallest – in terms of cardinality – subcover $\mathcal{S}$ and associate to $\mathcal{U}$ the finite quantity $\log_2(\mathrm{Card}(\mathcal{S}))$. This quantity, obviously, need not be finite in the general case of an arbitrary cover on a non-compact set.

However, a generalisation of entropy to non-compact sets can easily be defined by restricting the usual definition to *finite* covers[12]. This is the definition we will use here.

**Definition 34.** Let $\mathbf{X}$ be a topological space, and $\mathcal{U} = (U_i)_{i\in I}$ be a finite cover of $\mathbf{X}$. We define the quantity $H^0_{\mathbf{X}}(\mathcal{U})$ as

$$\min\{\log_2(\mathrm{Card}(J)) \mid J \subset I, \cup_{j\in J} U_j = \mathbf{X}\}.$$

In other words, if $k$ is the cardinality of the smallest subcover of $\mathcal{U}$, $H^0(\mathcal{U}) = \log_2(k)$.

[12]This is discussed by Hofer [36] together with another generalisation based on the Stone-Čech compactification of the underlying space.

**Definition 35.** Let $\mathbf{X}$ be a topological space and $f : \mathbf{X} \to \mathbf{X}$ be a continuous map. For any finite open cover $\mathcal{U}$ of $\mathbf{X}$, we define:

$$H^k_{\mathbf{X}}(f,\mathcal{U}) = \frac{1}{k} H^0_{\mathbf{X}}(\mathcal{U} \vee f^{-1}(\mathcal{U}) \vee \cdots \vee f^{-(k-1)}(\mathcal{U})).$$

One can show that the limit $\lim_{n\to\infty} H^n_{\mathbf{X}}(f,\mathcal{U})$ exists and is finite [35]; it will be noted $h(f,\mathcal{U})$. The topological entropy of $f$ is then defined as the supremum of these values, when $\mathcal{U}$ ranges over the set of all finite covers $\mathrm{FCov}(\mathbf{X})$.

**Definition 36.** Let $\mathbf{X}$ be a topological space and $f : \mathbf{X} \to \mathbf{X}$ be a continuous map. The *topological entropy* of $f$ is defined as $h(f) = \sup_{\mathcal{U}\in\mathrm{FCov}(\mathbf{X})} h(f,\mathcal{U})$.

*5.2. Graphings and Entropy*

We now need to define the entropy of *deterministic graphing*. As mentioned briefly already, deterministic graphings on a space $\mathbf{X}$ are in one-to-one correspondence with partial dynamical systems on $\mathbf{X}$. To convince oneself of this, it suffices to notice that any partial dynamical system can be represented as a graphing with a single edge, and that if the graphing $G$ is deterministic its edges can be glued together to define a partial continuous function $[G]$. Thus, we only need to extend the notion of entropy to partial maps, and we can then define the entropy of a graphing $G$ as the entropy of its corresponding map $[G]$.

Given a finite cover $\mathcal{U}$, the only issue with partial continuous maps is that $f^{-1}(\mathcal{U})$ is not in general a cover. Indeed, $\{f^{-1}(U) \mid U \in \mathcal{U}\}$ is a family of open sets by continuity of $f$ but the union $\cup_{U\in\mathcal{U}} f^{-1}(U)$ is a strict subspace of $\mathbf{X}$ (namely, the domain of $f$). It turns out the solution to this problem is quite simple: we notice that $f^{-1}(\mathcal{U})$ is a cover of $f^{-1}(\mathbf{X})$ and now work with covers of subspaces of $\mathbf{X}$. Indeed, $\mathcal{U} \vee f^{-1}(\mathcal{U})$ is itself a cover of $f^{-1}(\mathbf{X})$ and therefore the quantity $H^2_{\mathbf{X}}(f,\mathcal{U})$ can be defined as $(1/2)H^0_{f^{-1}(\mathbf{X})}(\mathcal{U} \vee f^{-1}(\mathcal{U}))$.

We now generalise this definition to arbitrary iterations of $f$ by extending Definitions 35 and 36 to partial maps as follows.

**Definition 37.** Let $\mathbf{X}$ be a topological space and $f : \mathbf{X} \to \mathbf{X}$ be a continuous partial map. For any finite open cover $\mathcal{U}$ of $\mathbf{X}$, we define:

$$H^k_{\mathbf{X}}(f,\mathcal{U}) = \frac{1}{k} H^0_{f^{-k+1}(\mathbf{X})}(\mathcal{U} \vee f^{-1}(\mathcal{U}) \vee \cdots \vee f^{-(k-1)}(\mathcal{U})).$$

The *entropy* of $f$ is then defined as $h(f) = \sup_{\mathcal{U}\in\mathrm{FCov}(\mathbf{X})} h(f,\mathcal{U})$, where $h(f,\mathcal{U})$ is again defined as the limit $\lim_{n\to\infty} H^n_{\mathbf{X}}(f,\mathcal{U})$ [36].

Now, let us consider the special case of a graphing $G$ with set of control states $S^G$. For an intuitive understanding, one can think of $G$ as the representation of a PRAM machine. We focus on the specific open cover indexed by the set of control states, i.e. $\mathcal{S} = (\mathbf{X} \times \{s\}_{s\in S^G})$, and call it *the states cover*. We will now show how the partial entropy $H^k(G,\mathcal{S})$ is related to the set of *admissible sequence of states*. Let us define those first.

**Definition 38.** Let $G$ be a graphing, with set of control states $S^G$. An admissible sequence of states is a sequence $\mathbf{s} = s_1 s_2 \dots s_n$ of elements of $S^G$ such that for all $i \in \{1, 2, \dots, n-1\}$ there exists a subset $C_i$ of $\mathbf{X}$ – i.e. a set of configurations – such that $G$ contains an edge from $C_i \times \{s_i\}$ to (a subspace of) $C_{i+1} \times \{s_{i+1}\}$ (with the convention that $C_n = \mathbf{X}$).

*Example* 39. As an example, let us consider the very simple graphing with four control states $a, b, c, d$ and edges from $\mathbf{X} \times \{a\}$ to $\mathbf{X} \times \{b\}$, from $\mathbf{X} \times \{b\}$ to $\mathbf{X} \times \{c\}$, from $\mathbf{X} \times \{c\}$ to $\mathbf{X} \times \{b\}$ and from $\mathbf{X} \times \{c\}$ to $\mathbf{X} \times \{d\}$. Then the sequences *abcd* and *abcbcbc* are admissible, but the sequences *aba*, *abcdd*, and *abcba* are not.

**Lemma 40.** *Let $G$ be a graphing, and $\mathcal{S}$ its states cover. Then for all integer $k$, the set $\mathrm{Adm}_k(G)$ of admissible sequences of states of length $k > 1$ is of cardinality $2^{k.H^k(G,\mathcal{S})}$.*

*Proof.* We show that the set $\mathrm{Adm}_k(G)$ of admissible sequences of states of length $k$ has the same cardinality as the smallest subcover of $\mathcal{S} \vee [G]^{-1}(\mathcal{S}) \vee \dots \vee [G]^{-(k-1)}(\mathcal{S}))$. Hence $H^k(G, \mathcal{S}) = \frac{1}{k} \log_2(\mathrm{Card}(\mathrm{Adm}_k(G)))$, which implies the result.

The proof is done by induction. As a base case, let us consider the set of $\mathrm{Adm}_2(G)$ of admissible sequences of states of length 2 and the open cover $\mathcal{V} = \mathcal{S} \vee [G]^{-1}(\mathcal{S})$ of $D = [G]^{-1}(\mathbf{X})$. An element of $\mathcal{V}$ is an intersection $\mathbf{X} \times \{s_1\} \cap [G]^{-1}(\mathbf{X} \times \{s_2\})$, and it is therefore equal to $C[s_1, s_2] \times \{s_1\}$ where $C[s_1, s_2] \subset \mathbf{X}$ is the set $\{x \in \mathbf{X} \mid [G](x, s_1) \in \mathbf{X} \times \{s_2\}\}$. This set is empty if and only if the sequence $s_1 s_2$ belongs to $\mathrm{Adm}_2(G)$. Moreover, given another sequence of states $s_1' s_2'$ (not necessarily admissible), the sets $C[s_1, s_2]$ and $C[s_1, s_2]$ are disjoint. Hence a set $C[s_1, s_2]$ is *removable from the cover* $\mathcal{V}$ if and only if the sequence $s_1 s_2$ is not admissible. This implies the result for $k = 2$.

The step for the induction is similar to the base case. It suffices to consider the partition $\mathcal{S}_k = \mathcal{S} \vee [G]^{-1}(\mathcal{S}) \vee \dots \vee [G]^{-(k-1)}(\mathcal{S}))$ as $\mathcal{S}_{k-1} \vee [G]^{-(k-1)}(\mathcal{S})$. By the same argument, one can show that elements of $\mathcal{S}_{k-1} \vee [G]^{-(k-1)}(\mathcal{S})$ are of the form $C[\mathbf{s} = (s_0 s_1 \dots s_{k-1}), s_k] \times \{s_1\}$ where $C[\mathbf{s}, s_k] \subset \mathbf{X}$ is the set $\{x \in \mathbf{X} \mid \forall i = 2, \dots, k, [G]^{i-1}(x, s_1) \in \mathbf{X} \times \{s_i\}\}$. Again, these sets $C[\mathbf{s}, s_k]$ are pairwise disjoint and empty if and only if the sequence $s_0 s_1 \dots s_{k-1}, s_k$ is not admissible. □

A tractable bound on the number of admissible sequences of states can be obtained by noticing that the sequence $H^k(G, \mathcal{S})$ is *sub-additive*, i.e. $H^{k+k'}(G, \mathcal{S}) \leqslant H^k(G, \mathcal{S}) + H^{k'}(G, \mathcal{S})$. A consequence of this is that $H^k(G, \mathcal{S}) \leqslant kH^1(G, \mathcal{S})$. Thus the number of admissible sequences of states of length $k$ is bounded by $2^{k^2 H^1(G,\mathcal{S})}$. We now study how the cardinality of admissible sequences can be related to the entropy of $G$.

**Lemma 41.** *For all $\epsilon > 0$, there exists an integer $N$ such that for all $k \geqslant N$, $H^k(G, \mathcal{U}) < h([G]) + \epsilon$.*

*Proof.* Let us fix some $\epsilon > 0$. Notice that if we let $H_k(G,\mathcal{U}) = H^0(\mathcal{U} \vee [G]^{-1}(\mathcal{U}) \vee \cdots \vee [G]^{-(k-1)}(\mathcal{U})))$, the sequence $H_k(U)$ satisfies $H_{k+l}(\mathcal{U}) \leqslant H_k(\mathcal{U}) + H_l(\mathcal{U})$. By Fekete's lemma on subadditive sequences, this implies that $\lim_{k\to\infty} H_k/k$ exists and is equal to $\inf_k H_k/k$. Thus $h([G],\mathcal{U}) = \inf_k H_k/k$.

Now, the entropy $h([G])$ is defined as $\sup_{\mathcal{U}} \lim_{k\to\infty} H_k(\mathcal{U})/k$. This then rewrites as $\sup_{\mathcal{U}} \inf_k H_k(\mathcal{U})/k$. We can conclude that $h([G]) \geqslant \inf_k H_k(\mathcal{U})/k$ for all finite open cover $\mathcal{U}$.

Since $\inf_k H_k(\mathcal{U})/k$ is the limit of the sequence $H_k/k$, there exists an integer $N$ such that for all $k \geqslant N$ the following inequality holds: $|H_k(\mathcal{U})/k - \inf_k H_k(\mathcal{U})/k| < \epsilon$, which rewrites as $H_k(\mathcal{U})/k - \inf_k H_k(\mathcal{U})/k < \epsilon$. From this we deduce $H_k(\mathcal{U})/k < h([G]) + \epsilon$, hence $H^k(G,\mathcal{U}) < h([G]) + \epsilon$ since $H^k(G,\mathcal{U}) = H_k(G,\mathcal{U})$. □

**Lemma 42.** *Let $G$ be a graphing, and let $c : k \mapsto \mathrm{Card}(\mathrm{Adm}_k(G))$. Then $c(k) = O(2^{k.h([G])})$ as $k$ goes to infinity.*

Lastly, we prove a result bounding the entropy of a map $\alpha(m) * \beta(m')$ in the CREW of AMCs. The result is essentially a consequence of the product rule ([35, Theorem 3], or [37]) stating that the entropy of a product $h(f \times g)$ is bounded above by the sum $h(f) + h(g)$.

**Lemma 43.** *Let $\alpha : \mathrm{M}\langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X} \times \mathbf{Y}$ and $\beta : \mathrm{M}\langle H, \mathrm{Q}\rangle \curvearrowright \mathbf{X} \times \mathbf{Z}$ be AMCs such that every non-central element of $\beta$ acts as the identity on $\mathbf{Z}$. Then for all $m \in \mathrm{M}\langle G, \mathrm{R}\rangle$ and $m' \in \mathrm{M}\langle H, \mathrm{Q}\rangle$, the entropy of $\alpha(m) * \beta(m')$ is bounded by the sum of the entropies of $\alpha(m)$ and $\beta(m')$:*

$$h(\alpha(m) * \beta(m') \leqslant h(\alpha(m)) + h(\beta(m')).$$

*Proof.* We show that the entropy of $\alpha(m) * \beta(m')$ is bounded by the entropy of $\alpha(m) \times \beta(m)$. The result then follows by the product rule [37]. We distinguish two cases: the first case is when one of $\alpha(m)$ or $\beta(m')$ is central, i.e. $\alpha(m); \pi_{\mathbf{X}} = \pi_{\mathbf{X}}$ or $\beta(m'); \pi_{\mathbf{X}} = \pi_{\mathbf{X}}$, the second case is when both $\alpha(m)$ and $\beta(m')$ act non-trivially on $\mathbf{X}$.

For the first case, we may consider that $\beta(m')$ is central without loss of generality. It is then of the form $\tilde{\beta} \times \mathrm{Id}_{\mathbf{X}}$ with $\tilde{\beta} : \mathbf{Z} \to \mathbf{Z}$, and the key observation is that

$$\alpha(m) * \beta(m') = \alpha(m) \times \tilde{\beta}$$

in this case. We now apply the product rule on both identities. From the first identity, we get

$$h(\beta(m')) = h(\tilde{\beta}) + h(\mathrm{Id}_{\mathbf{X}} = h(\tilde{\beta}),$$

since the entropy of the identity is equal to 0, and from the second identity, we get

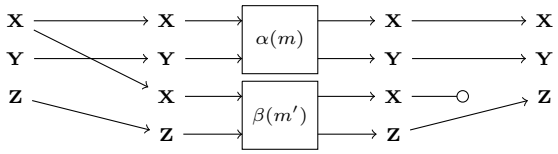
$$h(\alpha(m) * \beta(m')) \leq h(\alpha(m')) + h(\tilde{\beta}).$$

Combining both we obtain that $h(\alpha(m) * \beta(m')) = h(\alpha(m)) + h(\beta(m'))$.

For the second case, the definition of $\alpha(m) * \beta(m')$ states that it is equal to the following map:

$$\Delta_2; \alpha(m) \times (\beta(m'); \pi_{\mathbf{Z}}).$$

Diagrammatically, this is defined as:




We will now bound the entropy of $\alpha(m) * \beta(m')$. This is where we will use the hypothesis that $\beta(m')$ acts as the identity on $\mathbf{Z}$, i.e. that $\beta(m')(x, z) = (x', z)$. Indeed, from this hypothesis, one can deduce that

$$\alpha(m) * \beta(m')(x, y, z) = \alpha(m)(x, y) \times \mathrm{Id}_{\mathbf{Z}}(z),$$

hence

$$h(\alpha(m) * \beta(m')) = h(\alpha(m)) + h(\mathrm{Id}_{\mathbf{Z}}) = h(\alpha(m)) \leq h(\alpha(m)) + h(\beta(m')),$$

since $h(\mathrm{Id}_{\mathbf{Z}}) = 0$ and $h(\beta(m')) \geq 0$.

□

*5.3. Cells Decomposition*

Now, let us consider a deterministic graphing $G$, with its state cover $\mathcal{S}$. We fix a length $k > 2$ and reconsider the sets $C[\mathbf{s}] = C[(s_1 s_2 \dots s_{k-1}, s_k)]$ (for a sequence of states $\mathbf{s} = s_1 s_2 \dots s_k$) that appear in the proof of Lemma 40. The set $(C[\mathbf{s}])_{\mathbf{s} \in \mathrm{Adm}_k(G)}$ is a partition of the space $[G]^{-k+1}(\mathbf{X})$.

This decomposition splits the set of initial configurations into cells satisfying the following property: *for any two initial configurations contained in the same cell $C[\mathbf{s}]$, the $k$-th first iterations of $G$ goes through the same admissible sequence of states* $\mathbf{s}$.

**Definition 44.** Let $G$ be a deterministic graphing, with its state cover $\mathcal{S}$. Given an integer $k$, we define the $k$-fold decomposition of $\mathbf{X}$ along $G$ as the partition $\{C[\mathbf{s}] \mid \mathbf{s} \in \mathrm{Adm}_k(G)\}$.

Then Lemma 40 provides a bound on the cardinality of the $k$-th cell decomposition. Using the results in the previous section, we can then obtain the following proposition.

**Proposition 45.** *Let $G$ be a deterministic graphing, with entropy $h(G)$. The cardinality of the $k$-th cell decomposition of $\mathbf{X}$ w.r.t. $G$, as a function $c(k)$ of $k$, is asymptotically bounded by $g(k) = 2^{k.h([G])}$, i.e. $c(k) = O(g(k))$.*

We also state another bound on the number of cells of the $k$-th cell decomposition, based on the state cover entropy, i.e. the entropy with respect to the state cover rather than the usual entropy which takes the supremum of cover entropies when the cover ranges over all finite covers of the space. This result is a simple consequence of lemma 40.

**Proposition 2.** *Let $G$ be a deterministic graphing. We consider the* state cover entropy $h_0([G]) = \lim_{n \to \infty} H^n_{\mathbf{X}}([G], \mathcal{S})$ *where* $\mathbf{S}$ *is the state cover. The cardinality of the $k$-th cell decomposition of $\mathbf{X}$ w.r.t. $G$, as a function $c(k)$ of $k$, is asymptotically bounded by $g(k) = 2^{k.h_0([G])}$, i.e. $c(k) = O(g(k))$.*

## 6. First lower bounds

We will now explain how to obtain lower bounds for algebraic models of computation based on the interpretation of programs as graphings and entropic bounds. These results make use of the Milnor-Oleĭnik-Petrovskii-Thom theorem which bounds the sum of the Betti numbers of algebraic varieties. In fact, we will use a version due to Ben-Or of this theorem.

### *6.1. Milnor-Oleĭnik-Petrovskii-Thom theorem*

Let us first recall the classic Milnor-Oleĭnik-Petrovskii-Thom theorem. This theorem provides a bound on the sum of Betti numbers $\beta_i(V)$ of an algebraic variety $V$. Recall that the $i$-th Betti number is the dimension of the $i$-th Čech cohomology group $H_i(V)$; the number $\beta_0(V)$ then coincides with the number of connected components of the variety. In the following $H^*(V)$ denotes the sequence of cohomology groups, and $\operatorname{rank} H^*V$ should be understood as standing for the sum of all $\beta_i(V)$.

**Theorem 46** ([25, Theorem 3])**.** *If $V \subseteq \mathbf{R}^m$ is defined by polynomial identities of the form*

$$f_1 \geqslant 0, \ldots, f_p \geqslant 0$$

*with total degree $d = \deg f_1 + \cdots + \deg f_p$, then*

$$\operatorname{rank} H^*V \leqslant \frac{1}{2}(2+d)(1+d)^{m-1}.$$

We will use in the proof the following variant, stated and proved by Ben-Or.

**Theorem 47.** *Let $V \subseteq \mathbf{R}^n$ be a set defined by polynomial in·equations ($n, m, h \in \mathbf{N}$):*

$$\begin{cases} q_1(x_1, \ldots, x_n) = 0 \\ \vdots \\ q_m(x_1, \ldots, x_n) = 0 \\ p_1(x_1, \ldots, x_n) > 0 \\ \vdots \\ p_s(x_1, \ldots, x_n) > 0 \\ p_{s+1}(x_1, \ldots, x_n) \geqslant 0 \\ \vdots \\ p_h(x_1, \ldots, x_n) \geqslant 0 \end{cases}$$

*for $p_i, q_i \in \mathbf{R}[X_1, \ldots, X_n]$ of degree lesser than $d$.*

*Then $\beta_0(V)$ is at most $d(2d-1)^{n+h-1}$, where $d = \max\{2, \deg(q_i), \deg(p_j)\}$.*

*6.2. Algebraic decision trees*

From proposition 45, one obtains easily the following technical lemma.

**Lemma 48.** *Let $T$ be a $d$-th order algebraic decision tree deciding a subset $W \subseteq \mathbf{R}^n$. Then the number of connected components of $W$ is bounded by $2^h d(2d-1)^{n+h-1}$, where $h$ is the height of $T$.*

*Proof.* We let $h$ be the height of $T$, and $d$ be the maximal degree of the polynomials appearing in $T$. Then the $h$-th cell decomposition of $[T]$ defines a family of semi-algebraic sets defined by $h$ polynomials equalities and inequalities of degree at most $d$. By theorem 47, each of the cells have at most $d(2d-1)^{n+h-1}$ connected components. Moreover, proposition 2 states that this family has cardinality bounded by $2^{h.h_0([T])}$; since $h_0([T]) = 1$ because each state has at most one antecedent state, this bound becomes $2^h$. Thus, the $h$-th cell decomposition defines at most $2^h$ algebraic sets which have at most $d(2d-1)^{n+h-1}$ connected components. Since the set $W$ decided by $T$ is obtained as a union of the semi-algebraic sets in the $h$-th cell decomposition, it has at most $2^h d(2d-1)^{n+h-1}$ connected components. □

**Corollary 3** (Steele and Yao [1])**.** *A $d$-th order algebraic decision tree deciding a subset $W \subseteq \mathbf{R}^n$ with $N$ connected components has height $\Omega(\log N)$.*

This result of Steele and Yao adapts in a straightforward manner to a notion of algebraic computation trees describing the construction of the polynomials to be tested by mean of multiplications and additions of the coordinates. The authors remarked this result uses techniques quite similar to that of Mulmuley's lower bounds for the model of PRAMs *without bit operations*. It is also strongly similar to the techniques used by Cucker in proving that $\mathrm{NC}_{\mathbf{R}} \subsetneq \mathrm{PTIME}_{\mathbf{R}}$ [3].

However, a refinement of Steele and Yao's method was quickly obtained by Ben-Or so as to allow for computing divisions and taking square roots in this notion of algebraic computation trees. In the next section, we explain Ben-Or techniques from within the framework of graphings through the introduction of *entropic co-trees*. We first explain how the present approach already captures Mulmuley's proof of lower bounds for PRAMs *without bit operations*, which we will later strengthen using entropic co-trees.

*6.3. Mulmuley's result*

At this point, we are already capable of recovering Mulmuley's proof of lower bounds for PRAMs *without bit operations* [4]. The gist of the proof is to notice that given a PRAM $M$ over integers *not using* the instructions $//_{\cdot}(\cdot,\cdot)$ or $\sqrt[n]{\cdot}(\cdot)$, one can define a real-valued PRAM $\tilde{M}$ *not using* the instructions $/_{\cdot}(\cdot,\cdot)$ or $\sqrt[n]{\cdot}(\cdot)$ such that:

> $M$ accepts an integer-valued point $\vec{x}$ in $k$ steps if and only if $\tilde{M}$ accepts $\vec{x}$ in $k$ steps.

In particular, the subset $W \subseteq \mathbf{Z}^n$ decided by $M$ is contained in the subset $\bar{W} \subseteq \mathbf{R}^n$ decided by $\bar{M}$, and $W^c \subseteq \bar{W}^c$ – where the complement of $W$ is taken in $\mathbf{Z}^n$ and the complement of $\bar{W}$ is taken in $\mathbf{R}^n$. Moreover, the number of steps needed by $\bar{M}$ to decide if $\vec{x}$ belongs to $\bar{W}$ is equal to the number of steps needed by $M$ to decide if $\vec{x}$ belongs to $W$.

The proof of this is straightforward, as each instructions available in the integer-valued PRAMs model coincide with the restriction to integer values of an instruction available in the integer-valued PRAMs model. We can then prove the following result.

**Lemma 49.** *A set $V$ decided by a* PRAM without bit operations *$M$ with $p$ processors in $t$ steps is described by a set of in·equations of total degree*[13] *bounded by $pt.2^{O(pt)}$.*

*Proof.* Now, one supposes that $M$ computes some set $V$ in $t$ steps. Then it is computed by $\bar{M}$ in $t$ steps. Applying proposition 45 on the translation of $\bar{M}$ we obtain that the $t$-th cell decomposition of $[\![\bar{M}]\!]$ contains at most $2^{t.h_0([T])}$ cells. Moreover, each cell is described by a system of at most $pt$ polynomial in·equations of degree at most 2. Thus, the whole set decided by $\bar{M}$ is described by at most $pt.2^{t.h_0([T])}$ in·equations of degree at most 2. Lastly, one can use the fact that any non-central element in $\alpha_{\mathbf{R}\text{full}}$ act as the identity on the private memory[14] and lemma 43 to establish that $h_0([T])$ grows linearly w.r.t. the number of processors. □

This lemma is the first part of Mulmuley's proof of lower bounds. The second part of the proof, which does not differ from Mulmuley's argument, is detailed and reformulated in Section 9. We only provide a quick sketch of Mulmuley's result here, based on result presented in this later section. Note that the proof of the more general result Theorem 8 follows the same general pattern.

**Corollary 4.** *Let $M$ be a* PRAM *without bit operations, with at most $2^{O((\log N)^c)}$ processors, where $N$ is the length of the inputs and $c$ any positive integer.*

*Then $M$ does not decide* `maxflow` *in $O((\log N)^c)$ steps.*

*Sketch.* The crux is the obtention of Theorem 68 which, combined with Theorem 63, implies that there exists a polynomial $P$ such that no surface of total degree $\delta$ can separate the integer points defined by `maxflow` as long as $2^{\Omega(N)} > P(\delta)$. Hence, if we suppose that $M$ is such that both the running time $t$ and number of processors $p$ are $O((\log^k(N))^c)$ (for any positive integers $k$ and $c$), the previous result implies that the total degree $\delta$ of the set decided by $\bar{M}$ is at most $O(2^{pt})$. Hence $2^{\Omega(N)} > P(\delta)$ for any polynomial $P$, and we conclude that $\bar{M}$ does not compute a set separating the integer points defined by `maxflow`, hence $M$ does not decide `maxflow`. □

[13] The sum of the degrees of the polynomials defining $V$.

[14] The only non-central elements are those modifying the shared memory, and those do not modify the private memory.

## 7. Refining the method

It is not a surprise then that similar bounds to that of algebraic decisions trees can be computed using similar methods in the restricted fragment without division and square roots. An improvement on this is the result of Ben-Or generalising the technique to algebraic computation trees with division and square root nodes. The principle is quite simple: one simply adds additional variables to avoid using the square root or division, obtaining in this way a system of polynomial equations. For instance, instead of writing the equation $p/q < 0$, one defines a fresh variable $r$ and considers the system

$$p = qr; r < 0$$

This method seems different from the direct entropy bound obtained in the case of algebraic decision trees. However, we will see how it can be adapted directly to graphings.

### *7.1. Entropic co-trees and k-th computational forests*

**Definition 50** ($k$-th entropic co-tree)**.** Consider a deterministic graphing representative $T$, and fix an element $\top$ of the set of control states. We can define the $k$-th entropic co-tree of $T$ along $\top$ and the state cover inductively:

- $k = 0$, the co-tree $\mathrm{coT}_0(T)$ is simply the root $n^\epsilon = \mathbf{R}^n \times \{\top\}$;
- $k = 1$, one considers the preimage of $n^\epsilon$ through $T$, i.e. $T^{-1}(\mathbf{R}^n \times \{\top\})$ the set of all non-empty sets $\alpha(m_e)^{-1}(\mathbf{R}^n \times \{\top\})$ and intersects it pairwise with the state cover, leading to a finite family (of cardinality bounded by the number of states multiplied by the number of edges fo $T$) $(n_e^i)_i$ defined as $n^i = T^{-1}(n^\epsilon) \cap \mathbf{R}^n \times \{i\}$. The first entropic co-tree $\mathrm{coT}_1(T)$ of $T$ is then the tree defined by linking each $n_e^i$ to $n^\epsilon$ with an edge labelled by $m_e$;
- $k + 1$, suppose defined the $k$-th entropic co-tree of $T$, defined as a family of elements $n_{\mathbf{e}}^\pi$ where $\pi$ is a finite sequence of states of length at most $k$ and $\mathbf{e}$ a sequence of edges of $T$ of the same length, and where $n_{\mathbf{e}}^\pi$ and $n_{\mathbf{e}'}^{\pi'}$ are linked by an edge labelled $f$ if and only if $\pi' = \pi.s$ and $\mathbf{e}' = f.\mathbf{e}$ where $s$ is a state and $f$ an edge of $T$. We consider the subset of elements $n_{\mathbf{e}'}^\pi$ where $\pi$ is exactly of length $k$, and for each such element we define new nodes $n_{e.\mathbf{e}'}^{\pi.s}$ defined as $\alpha(m_e)^{-1}(n_{\mathbf{e}'}^\pi) \cap \mathbf{R}^n \times \{s\}$ when it is non-empty. The $k+1$-th entropic co-tree $\mathrm{coT}_{k+1}(T)$ is defined by extending the $k$-th entropic co-tree $\mathrm{coT}_k(T)$, adding the nodes $n_{e.\mathbf{e}'}^{\pi.s}$ and linking them to $n_{\mathbf{e}'}^\pi$ with an edge labelled by $e$.

*Remark.* The co-tree can alternatively be defined non-inductively in the following way: the $n_{\mathbf{e}}^\pi$ for $\pi$ is a finite sequence of states and $\mathbf{e}$ a sequence of edges of $T$ of the same length by $n_\epsilon^\epsilon = \mathbf{R}^n \times \{\top\}$ and

$$n_{\mathbf{e}.e}^{\pi.s} = \left[\alpha(m_e)^{-1}(n_{\mathbf{e}}^\pi)\right] \cap \left[\mathbf{R}^n \times \{s\}\right]$$

The $k$-th entropic co-tree of $T$ along $\top$ has as vertices the non-empty sets $n_{\mathbf{e}}^\pi$ for $\pi$ and $\mathbf{e}$ of length at most $k$ and as only edges, links $n_{\mathbf{e}.e}^{\pi.s} \to n_{\mathbf{e}}^\pi$ labelled by $m_e$.

This definition formalises a notion that appears more or less clearly in the work of Lipton and Steele, and of Ben-Or, as well as in the proof by Mulmuley. The nodes for paths of length $k$ in the $k$-th co-tree corresponds to the $k$-th cell decomposition, and the corresponding path defines the polynomials describing the semi-algebraic set decided by a computational tree. The co-tree can be used to reconstruct the algebraic computation tree $T$ from the graphing representative $[T]$, or constructs *some* algebraic computation tree (actually a forest) that approximates the computation of the graphing $F$ under study when the latter is not equal to $[T]$ for some tree $T$.

**Definition 51** ($k$-th computational forest)**.** Consider a deterministic graphing $T$, and fix an element $\top$ of the set of control states. We define the $k$-th computational forest of $T$ along $\top$ and the state cover as follows. Let $\mathrm{coT}_k(T)$ be the $k$-th entropic co-tree of $T$. The $k$-th computational forest of $T$ is defined by regrouping all elements $n^{\pi}_{e.\vec{e'}}$ of length $m$: if the set $N^m_e = \{n^{\pi}_{e.\vec{e'}} \in \mathrm{coT}_k(T) \mid \mathrm{len}(\pi) = m\}$ is non-empty it defines a new node $N^m_e$. Then one writes down an edge from $N^m_e$ to $N^{m-1}_{e'}$, labelled by $e$, if and only if there exists $n^{s.\pi}_{e.e'.\vec{f}} \in N^m_e$ such that $n^{\pi}_{e'.\vec{f}} \in N^{m-1}_{e'}$.

One checks easily that the $k$-th computational forest is indeed a forest: an edge can exist between $N^m_e$ and $N^n_f$ only when $n = m + 1$, a property that forbids cycles. The following proposition shows how the $k$-th computational forest is linked to computational trees.

**Proposition 52.** *If $T$ is a computational tree of depth $k$, the $k$-th computational forest of $[T]$ is a tree which defines straightforwardly a graphing (treeing) representative of $T$.*

We now state and prove an easy bound on the size of the entropic co-trees.

**Proposition 53** (Size of the entropic co-trees)**.** *Let $T$ be a graphing representative, $E$ its set of edges, and $\mathrm{Seq}_k(E)$ the set of paths of length $k$ in $T$. The number of nodes of its $k$-th entropic co-tree $\mathrm{coT}_k(T)$, as a function $n(k)$ of $k$, is asymptotically bounded by $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{(k+1).h([G])}$, itself bounded by $2^{\mathrm{Card}(E)}.2^{(k+1).h([G])}$.*

*Proof.* For a fixed sequence $\vec{e}$, the number of elements $n^{\pi}_{\vec{e}}$ of length $m$ in $\mathrm{coT}_k(T)$ is bounded by the number of elements in the $m$-th cell decomposition of $T$, and is therefore bounded by $g(m) = 2^{m.h([T])}$ by proposition 45. The number of sequences $\vec{e}$ is bounded by $\mathrm{Card}(\mathrm{Seq}_k(E))$ and therefore the size of $\mathrm{coT}_k(T)$ is thus bounded by $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{(k+1).h([T])}$. □

From the proof, one sees that the following variant of proposition 2 holds.

**Proposition 54.** *Let $G$ be a deterministic graphing with a finite set of edges $E$, and $\mathrm{Seq}_k(E)$ the set of paths of length $k$ in $G$. We consider the* state cover entropy *$h_0([G]) = \lim_{n\to\infty} H^n_{\mathbf{X}}([G], \mathcal{S})$ where $\mathbf{S}$ is the state cover. The cardinality of the length $k$ nodes of the entropic co-tree of $G$, as a function $c(k)$ of $k$, is asymptotically bounded by $g(k) = \mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$, which is itself bounded by $2^{\mathrm{Card}}(E).2^{k.h_0([G])}$.*

*7.2. The technical lemma*

This definition formalises a notion that appears more or less clearly in the work of Steele and Yao, and of Ben-Or, as well as in the proof by Mulmuley. It will be key in establishing the main technical lemma, namely lemma 5.

The vertices for paths of length $k$ in the $k$-th co-tree corresponds to the $k$-th cell decomposition, and the corresponding path defines the polynomials describing the semi-algebraic set decided by a computational tree. While in Steele and Yao and Mulmuley's proofs, one obtain directly a polynomial for each cell, we here need to construct a system of equations for each branch of the co-tree.

Given a $\textsc{crew}^p(\alpha_{\mathbf{R}\text{full}})$-graphing representative $G$ we will write $\sqrt[\partial]{G}$ the maximal value of $n$ for which an instruction $\sqrt[n]{i}(j)$ appears in the realiser of an edge of $G$.

The proof of this theorem is long but simple to understand as it follows Ben-Or's method. We define, for each vertex of the $k$-th entropic co-tree, a system of algebraic equations (each of degree at most 2). The system is defined by induction on $k$, and uses the information of the specific instruction used to extend the sequence indexing the vertex at each step. For instance, the case of division follows Ben-Or's method, introducing a fresh variable and writing down two equations. As mentioned in footnote 8, the input variables are split into numerical and non-numerical inputs, and one assumes that indirect references do not depend on non-numerical inputs. This implies that all indirect references have a fixed value determined by the non-numerical input; hence in the analysis below – which focuses on numerical inputs – indirect references correspond to references to a fixed value register.

**Lemma 55.** *Let $G$ be a computational graphing representative with edges realised only by generators of the AMC $\textsc{crew}^p(\alpha_{\mathbf{R}\text{full}})$, and $\mathrm{Seq}_k(E)$ the set of paths of length $k$ in $G$. Suppose $G$ computes the membership problem for $W \subseteq \mathbf{R}^n$ in $k$ steps, i.e. for each element of $\mathbf{R}^n$, $\pi_{\mathbf{S}}(G^k(x)) = \top$ if and only if $x \in W$. Then $W$ is a semi-algebraic set defined by at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ systems of $pk$ equations of degree at most $\max(2, \sqrt[\partial]{G})$ and involving at most $p(k+n)$ variables and $p(k+n)$ inequalities.*

*Proof.* If $G$ computes the membership problem for $W$ in $k$ steps, it means $W$ can be described as the union of the subspaces corresponding to the nodes $n^{\pi}_{\mathbf{e}}$ with $\pi$ of length $k$ in $\textsc{cot}_k(T)$. Now, each such subspace is an algebraic set, as it can be described by a set of polynomials as follows.

Finally let us note that, as in Mulmuley's work [4], since in our model the memory pointers are allowed to depend only on the nonnumeric parameters, indirect memory instructions can be treated as standard – direct – memory instructions. In other words, whenever an instruction involving a memory pointer is encountered during the course of execution, the value of the pointer is completely determined by nonnumerical data, and the index of the involved registers is completely determined, independently of the numerical inputs.

We define a system of equations $(E^{\mathbf{e}}_i)_i$ for each node $n^{\pi}_{\mathbf{e}}$ of the entropic co-tree $\textsc{cot}_k(T)$. We explicit the construction for the case $p = 1$, i.e. for the

AMC CREW$^1(\alpha_{\mathbf{R}\text{full}}) = \alpha_{\mathbf{R}\text{full}}$; the case for arbitrary $p$ is then dealt with by following the construction and introducing $p$ equations at each step (one for each of the $p$ instructions in $\alpha_{\mathbf{R}\text{full}}$ corresponding to an element of CREW$^p(\alpha_{\mathbf{R}\text{full}})$). This is done inductively on the size of the path $\vec{e}$, keeping track of the last modifications of each register. I.e. we define both the system of equations $(E_i^{\mathbf{e}})_i$ and a function[15] $\mathfrak{h}(\mathbf{e}) : \mathbf{R}^\omega \cup \{\perp\} \to \omega$ (which is almost everywhere null). This function increases each time a register is modified, and will be used to create a new variable corresponding to the value of the register *at this precise moment in the computation*. The additional value $\perp$ will be used to create new variables not related to a specific register (used in the case of comparisons below).

For an empty sequence, the system of equations is empty, and the function $\mathfrak{h}(\epsilon)$ is constant, equal to 0. The system of equation, as well as the function $\mathfrak{h}(\epsilon)$, are then jointly defined inductively as follows. Suppose that $\vec{e'} = (e_1, \ldots, e_m, e_{m+1})$, with $\vec{e} = (e_1, \ldots, e_m)$, and that one already computed $(E_i^{\mathbf{e}})_{i \geqslant m}$ and the function $\mathfrak{h}(\mathbf{e})$. We now consider the edge $e_{m+1}$ and let $(r, r')$ be its realizer. We extend the system of equations $(E_i^{\mathbf{e}})_{i \geqslant m}$ by a new equation $E_{m+1}$ and define the function $\mathfrak{h}(\mathbf{e}')$ as follows:

- if $r = +_i(j,k)$, $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{i,\mathfrak{h}(\mathbf{e}')(i)} = x_{j,\mathfrak{h}(\mathbf{e}')(j)} + x_{k,\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = -_i(j,k)$, $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{i,\mathfrak{h}(\mathbf{e}')(i)} = x_{j,\mathfrak{h}(\mathbf{e}')(j)} - x_{k,\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = \times_i(j,k)$, $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{i,\mathfrak{h}(\mathbf{e}')(i)} = x_{j,\mathfrak{h}(\mathbf{e}')(j)} \times x_{k,\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = /_i(j,k)$, $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{i,\mathfrak{h}(\mathbf{e}')(i)} \times x_{k,\mathfrak{h}(\mathbf{e}')(k)} = x_{j,\mathfrak{h}(\mathbf{e}')(j)}$;
- if $r = +_i^c(k)$, $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{i,\mathfrak{h}(\mathbf{e}')(i)} = c + x_{k,\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = -_i^c(k)$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{i,\mathfrak{h}(\mathbf{e}')(i)} = c - x_{k,\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = \times_i^c(k)$, $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{i,\mathfrak{h}(\mathbf{e}')(i)} = c \times x_{k,\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = /_i^c(k)$, $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{i,\mathfrak{h}(\mathbf{e}')(i)} \times c = x_{k,\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = \sqrt[n]{i}(k)$, $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $(x_{i,\mathfrak{h}(\mathbf{e}')(i)})^n = x_{k,\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = \mathtt{copy}(i,j)$, $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{i,\mathfrak{h}(\mathbf{e}')(i)} = x_{j,\mathfrak{h}(\mathbf{e}')(j)}$;

[15] The use of $\perp$ is to allow for the creation of fresh variables not related to a register.

- if $r = \mathtt{copy}(\sharp i, j)$, then the value of $\sharp i$ does not depend on the numerical inputs and corresponds to a fixed value $a \in \mathbf{R}$; we then define $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = a$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{a,\mathfrak{h}(\mathbf{e}')(a)} = x_{j,\mathfrak{h}(\mathbf{e}')(j)}$;
- if $r = \mathtt{copy}(i, \sharp j)$, then the value of $\sharp j$ does not depend on the numerical inputs and corresponds to a fixed value $a \in \mathbf{R}$; we then define $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = i$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise; then $E_{m+1}$ is $x_{i,\mathfrak{h}(\mathbf{e}')(i)} = x_{a,\mathfrak{h}(\mathbf{e}')(a)}$;
- if $r = \mathrm{Id}$, the source of the edge $e_q$ is of the form $\{(x_1, \ldots, x_{n+\ell}) \in \mathbf{R}^{n+\ell} \mid P(x_k)\} \times \{i\}$ where $P$ compares the variable $x_k$ with 0:
  - if $P(x_k)$ is $x_k \neq 0$, $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u) + 1$ if $u = \perp$, and $\mathfrak{h}(\mathbf{e}')(u) = \mathfrak{h}(\mathbf{e})(u)$ otherwise then $E_{m+1}$ is $x_{\perp,\mathfrak{h}(\mathbf{e}')(\perp)} x_{k,\mathfrak{h}(\mathbf{e}')(k)} - 1 = 0$;
  - otherwise (e.g. $P(x_k)$ is the inequality $x_k \leq 0$) we set $\mathfrak{h}(\mathbf{e}') = \mathfrak{h}(\mathbf{e})$ and $E_{m+1}$ is defined as $P(x_{k,\mathfrak{h}(\mathbf{e}')(k)})$.

We now consider the system of equations $(E_i)_{i=1}^k$ defined from the path $\mathbf{e}$ of length $k$ corresponding to a node $n_{\mathbf{e}}^{\pi}$ of the $k$-th entropic co-tree of $G$. This system consists in $k$ equations of degree at most $\max(2, \sqrt[\partial]{G})$ and containing at most $k + n$ variables, counting the variables $x_1^0, \ldots, x_n^0$ corresponding to the initial registers, and adding at most $k$ additional variables since an edge of $\vec{e}$ introduces at most one fresh variable. Among these equations, at most $k$ are inequalities, since each edge introduces at most one inequation. Since the number of vertices $n_{\mathbf{e}}^{\pi}$ is bounded by $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ by proposition 54, we obtained the stated result in the case $p = 1$.

The case for arbitrary $p$ is then deduced by noticing that each step in the induction would introduce at most $p$ new equations and $p$ new variables. The resulting system thus contains at most $pk$ equations of degree at most $\max(2, \sqrt[\partial]{G})$ and containing at most $p(k + n)$ variables. □

This theorem extends to the case of general computational graphings by considering the *algebraic degree* of the graphing.

**Definition 56** (Algebraic degree)**.** Let $\alpha : \langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}$ be an AMC. The algebraic degree of an element of $\mathrm{M}\langle G, \mathrm{R}\rangle$ is the minimal number of generators needed to express it. The algebraic degree of an $\alpha$-graphing is the maximum of the algebraic degrees of the realisers of its edges.

If an edge is realised by an element $m$ of algebraic degree $D$, then the method above applies by introducing the $D$ new equations corresponding to the $D$ generators used to define $m$. The general result then follows.

**Lemma 5.** *Let $G$ be a CREW$^p(\alpha_{\mathbf{R}\mathrm{full}})$-computational graphing representative, $\mathrm{Seq}_k(E)$ the set of paths of length $k$ in $G$, and $D$ its algebraic degree. Suppose $G$ computes the membership problem for $W \subseteq \mathbf{R}^n$ in $k$ steps, i.e. for each element of $\mathbf{R}^n$, $\pi_{\mathbf{S}}(G^k(x)) = \top$ if and only if $x \in W$. Then $W$ is a semi-algebraic set defined by at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ systems of $pkD$ equations of degree at most $\max(2, \sqrt[\partial]{G})$ and involving at most $pD(k + n)$ variables.*

## 8. Recovering Ben Or and Cucker's theorems

*8.1. Ben-Or*

We now recover Ben-Or result by obtaining a bound on the number of connected components of the subsets $W \subseteq \mathbf{R}^n$ whose membership problem is computed by a graphing in less than a given number of iterations. This theorem is obtained by applying the Milnor-Oleĭnik-Petrovskii-Thom theorem on the obtained systems of equations to bound the number of connected components of each cell. Notice that in this case $p = 1$ and $\sqrt[\partial]{G} = 2$ since the model of algebraic computation trees use only square roots. A mode general result holds for algebraic computation trees extended with arbitrary roots, but we here limit ourselves here to the original model.

**Theorem 57.** *Let $G$ be a computational $\alpha_{\mathbf{R}\text{full}}$-graphing representative translating an algebraic computational tree, $\mathrm{Seq}_k(E)$ the set of length $k$ sequences of edges in $G$. Suppose $G$ computes the membership problem for $W \subseteq \mathbf{R}^n$ in $k$ steps. Then $W$ has at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])+1}3^{2k+n-1}$ connected components.*

*Proof.* By Lemma 55 (using the fact that $p = 1$ and $\sqrt[\partial]{G} = 2$), the problem $W$ decided by $G$ in $k$ steps is described by at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ systems of $k$ equations of degree 2 involving at most $k+n$ variables and at most $k$ inequalities. Applying Theorem 47, we deduce that each such system of in·equations (of $k$ equations of degree 2 in $\mathbf{R}^{k+n}$) describes a semi-algebraic variety $S$ such that $\beta_0(S) < 2.3^{(n+k)+k-1}$. This being true for each of the $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ cells, we have that $\beta_0(W) < \mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])+1}3^{2k+n-1}$. □

Since a subset computed by a tree $T$ of depth $k$ is computed by $[\![T]\!]$ in $k$ steps by Theorem 19, we get as a corollary the original theorem by Ben-Or relating the number of connected components of a set $W$ and the depth of the algebraic computational trees that compute the membership problem for $W$.

**Corollary 6** ([2, Theorem 5])**.** *Let $W \subseteq \mathbf{R}^n$ be any set, and let $N$ be the maximum of the number of connected components of $W$ and $\mathbf{R}^n \setminus W$. An algebraic computation tree computing the membership problem for $W$ has height $\Omega(\log N)$.*

*Proof.* Let $T$ be an algebraic computation tree computing the membership problem for $W$, and consider the computational treeing $[T]$. Let $d$ be the height of $T$; by definition of $[T]$ the membership problem for $W$ is computed in exactly $d$ steps. Thus, by the previous theorem, $W$ has at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{d.h_0([T])+1}3^{2d+n-1}$ connected components. As the interpretation of an algebraic computational tree, $h_0([T])$ is at most equal to 2, and $\mathrm{Card}(\mathrm{Seq}_k(E))$ is bounded by $2^d$. Hence $N \leqslant 2^d.2^{2d+1}3^{n-1}3^{2d}$, i.e. $d = \Omega(\log N)$. □

We immediately deduce an application that will be useful to us in the remainder. Let $m \in \mathbf{N}$ and $0 < x < 2^m$. Let $k \in \mathbf{N}$ be such that $1 \leqslant k \leqslant m$. We call $\left\lfloor \frac{x}{2^{k-1}} \right\rfloor - 2 \left\lfloor \frac{x}{2^k} \right\rfloor$ the *k-th bit* of $x$.

**Lemma 58.** *An algebraic computation tree computing the k-th bit of x has height* $\Omega(m-k)$.

*Proof.* Let

$$W = \left\{ x \in \mathbf{R} \mid \left\lfloor \frac{x}{2^{k-1}} \right\rfloor - 2 \left\lfloor \frac{x}{2^k} \right\rfloor = 1 \right\}$$

$W$ is the disjoint union of $2^{m-k+1}$ intervals, and so is its complement in $]0; 2^m[$. So, by Theorem 6, any algebraic computation tree computing the $k$-th bit has height $\Omega(m-k)$. □

We will see later that bit-extraction is also difficult for the PRAM model (cf. Prop. **??**). This is an essential difference between the booleans and algebraic models.

*Remark.* In the case of algebraic PRAMs discussed in the next sections, the $k$-th entropic co-tree $\text{coT}_k(T)[M]$ of a machine $M$ defines an algebraic computation tree which follows the $k$-th first steps of computation of $M$. I.e. the algebraic computation tree $\text{coT}_k(T)[M]$ approximate the computation of $M$ in such a way that $M$ and $\text{coT}_k(T)[M]$ behave in the exact same manner in the first $k$ steps.

*8.2. Cucker's theorem*

Cucker's proof considers the problem defined as the following algebraic set.

**Definition 59.** Define $\mathfrak{Fer}$ to be the set:

$$\{x \in \mathbf{R}^\omega \mid |x| = n \Rightarrow x_1^{2^n} + x_2^{2^n} = 1\},$$

where $|x| = \max\{n \in \omega \mid x_n \neq 0\}$.

It can be shown to lie within $\text{PTIME}_{\mathbf{R}}$, i.e. it is decided by a real Turing machine [38] – i.e. working with real numbers and real operations –, running in polynomial time.

**Theorem 60** (Cucker ([3], Proposition 3))**.** *The problem* $\mathfrak{Fer}$ *belongs to* $\text{PTIME}_{\mathbf{R}}$.

We now prove that $\mathfrak{Fer}$ is not computable by an algebraic circuit of polylogarithmic depth. The proof follows Cucker's argument, but uses the lemma proved in the previous section.

**Corollary 7** (Cucker ([3], Theorem 3.2))**.** *No algebraic circuit of depth* $k = \log^i n$ *and size*[16] *kp compute* $\mathfrak{Fer}$.

[16] We notice here that we do not assume any bounds on the number of processors.

*Proof.* For this, we will use the lower bounds result obtained in the previous section. Indeed, by Theorem 22 and Lemma 5, any problem decided by an algebraic circuit of depth $k$ is a semi-algebraic set defined by at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ systems of $k$ equations of degree at most $\max(2, \sqrt[\partial]{G}) = 2$ (since only square roots are allowed in the model) and involving at most $k+n$ variables. But the curve $\mathfrak{F}^{\mathbf{R}}_{2^n}$ defined as $\{x_1^{2^n} + x_2^{2^n} - 1 = 0 \mid x_1, x_2 \in \mathbf{R}\}$ is infinite. As a consequence, one of the systems of equation must describe a set containing an infinite number of points of $\mathfrak{F}^{\mathbf{R}}_{2^n}$.

This set $S$ is characterized, up to some transformations on the set of equations obtained from the entropic co-tree, by a finite system of inequalities of the form

$$\bigwedge_{i=1}^{s} F_i(X_1, X_2) = 0 \wedge \bigwedge_{j=1}^{t} G_j(X_1, X_2) < 0,$$

where $t$ is bounded by $kp$ and the degree of the polynomials $F_i$ and $G_i$ are bounded by $2^k$. Moreover, since $\mathfrak{F}^{\mathbf{R}}_{2^n}$ is a curve and no points in $S$ must lie outside of it, we must have $s > 0$.

Finally, the polynomials $F_i$ vanish on that infinite subset of the curve and thus in a 1-dimensional component of the curve. Since the curve is an irreducible one, this implies that every $F_i$ must vanish on the whole curve. Using the fact that the ideal $(X_1^{2^n} + X_2^{2^n} - 1)$ is prime (and thus radical), we conclude that all the $F_i$ are multiples of $X_1^{2^n} + X_2^{2^n} - 1$ which is impossible if their degree is bounded by $2^{\log^i n}$ as it is strictly smaller than $2^n$. □

## 9. Algebraic surfaces for an optimization problem

### *9.1. Geometric Interpretation of Optimization Problems*

We start by showing how decision problems of a particular form induce a binary partition of the space $\mathbf{Z}^d$: the points that are accepted and those that are rejected. Intuitively, the machine decides the problem if the partition it induces refines the one of the problem.

We will consider problems of a very specific form: decisions problems in $\mathbf{Z}^3$ associated to optimization problems. Let $\mathcal{P}_{\mathrm{opt}}$ be an optimization problem on $\mathbf{R}^d$. Solving $\mathcal{P}_{\mathrm{opt}}$ on an instance $t$ amounts to optimizing a function $f_t(\cdot)$ over a space of parameters. We note $\mathrm{Max}\mathcal{P}_{\mathrm{opt}}(t)$ this optimal value. An affine function $\mathsf{Param} : [p; q] \to \mathbf{R}^d$ is called a *parametrization* of $\mathcal{P}_{\mathrm{opt}}$. Such a parametrization defines naturally a decision problem $\mathcal{P}_{\mathrm{dec}}$: for all $(x, y, z) \in \mathbf{Z}^3$, $(x, y, z) \in \mathcal{P}_{\mathrm{dec}}$ iff $z > 0$, $x/z \in [p; q]$ and $y/z \leq \mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(x/z)$.

In order to study the geometry of $\mathcal{P}_{\mathrm{dec}}$ in a way that makes its connection with $\mathcal{P}_{\mathrm{opt}}$ clear, we consider the ambient space to be $\mathbf{R}^3$, and we define the *ray* $[p]$ of a point $p$ as the half-line starting at the origin and containing $p$. The projection $\Pi(p)$ of a point $p$ on a plane is the intersection of $[p]$ and the affine plane $\mathcal{A}_1$ of equation $z = 1$. For any point $p \in \mathcal{A}_1$, and all $p_1 \in [p]$, $\Pi(p_1) = p$. It is clear that for $(p, p', q) \in \mathbf{Z}^2 \times \mathbf{N}^+$, $\Pi((p, p', q)) = (p/q, p'/q, 1)$.

The *cone* $[C]$ of a curve $C$ is the set of rays of points of the curve. The projection $\Pi(C)$ of a surface or a curve $C$ is the set of projections of points in $C$. We note $\mathsf{Front}$ the frontier set

$$\mathsf{Front} = \{(x, y, 1) \in \mathbf{R}^3 \mid y = \mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(x)\}.$$

and we remark that

$$[\mathsf{Front}] = \{(x, y, z) \in \mathbf{R}^2 \times \mathbf{R}^+ \mid y/z = \mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(x/z)\}.$$

Finally, a machine $M$ decides the problem $\mathcal{P}_{\mathrm{dec}}$ if the sub-partition of accepting cells in $\mathbf{Z}^3$ induced by the machine is finer than the one defined by the problem's frontier $[\mathsf{Front}]$ (which is defined by the equation $y/z \leq \mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(x/z)$).

*9.2. Parametric Complexity*

We now further restrict the class of problems we are interested in: we will only consider $\mathcal{P}_{\mathrm{opt}}$ such that $\mathsf{Front}$ is simple enough. Precisely:

**Definition 61.** We say that $\mathsf{Param}$ is an *affine parametrization* of $\mathcal{P}_{\mathrm{opt}}$ if $\mathsf{Param}; \mathrm{Max}\mathcal{P}_{\mathrm{opt}}$ is

- convex
- piecewise linear, with breakpoints $\lambda_1 < \cdots < \lambda_\rho$
- such that the $(\lambda_i)_i$ and the $(\mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(\lambda_i))_i$ are all rational.

The *(parametric) complexity* $\rho(\mathsf{Param})$ is defined as the number of breakpoints of $\mathsf{Param}; \mathrm{Max}\mathcal{P}_{\mathrm{opt}}$.

An optimization problem that admits an affine parametrization of complexity $\rho$ is thus represented by a surface $[\mathsf{Front}]$ that is quite simple: the cone of the graph of a piecewise affine function, constituted of $\rho$ segments. We say that such a surface is a *$\rho$-fan*. This restriction seems quite serious when viewed geometrically. Nonetheless, many optimization problems admit such a parametrization. Before giving examples, we introduce another measure of the complexity of a parametrization.

**Definition 62.** Let $\mathcal{P}_{\mathrm{opt}}$ be an optimization problem and $\mathsf{Param}$ be an affine parametrization of it. The *bitsize* of the parametrization is the maximum of the bitsizes of the numerators and denominators of the coordinates of the breakpoints of $\mathsf{Param}; \mathrm{Max}\mathcal{P}_{\mathrm{opt}}$.

In the same way, we say that a $\rho$-fan is of *bitsize* $\beta$ if all its breakpoints are rational and the bitsize of their coordinates is lesser thant $\beta$.

**Theorem 63** (Murty [39], Carstensen [33])**.**

1. *there exists an affine parametrization of bitsize $O(n)$ and complexity $2^{\Omega(n)}$ of combinatorial linear programming, where $n$ is the total number of variables and constraints of the problem.*

2. *there exists an affine parametrization of bitsize $O(n^2)$ and complexity $2^{\Omega(n)}$ of the* `maxflow` *problem for directed and undirected networks, where $n$ is the number of nodes in the network.*

We refer the reader to Mulmuley's paper [4, Thm. 3.1.3] for proofs, discussions and references.

### *9.3. Algebraic Surfaces*

An algebraic surface in $\mathbf{R}^3$ is a surface defined by an equation of the form $p(x, y, z) = 0$ where $p$ is a polynomial. If $S$ is a set of surfaces, each defined by a polynomial, the *total degree* of $S$ is defined as the sum of the degrees of polynomials defining the surfaces in $S$.

Let $K$ be a compact of $\mathbf{R}^3$ delimited by algebraic surfaces and $S$ be a finite set of algebraic surfaces, of total degree $\delta$. We can assume that $K$ is actually delimited by two affine planes of equation $z = \mu$ and $z = 2\mu_z$ and the cone of a rectangle $\{(x, y, 1) \mid |x|, |y| \leqslant \mu_{x,y}\}$, by taking any such compact containing $K$ and adding the surfaces bounding $K$ to $S$. $S$ defines a partition of $K$ by considering maximal compact subspaces of $K$ whose boundaries are included in surfaces of $S$. Such elements are called the *cells* of the decomposition associated to $S$.

The cell of this partition can have complicated shapes: in particular, a cell can have a arbitrarily high number of surfaces of $S$ as boundaries. We are going to refine this partition into a partition $\mathrm{Col}_S$ whose cells are all bounded by cones of curves and at most two surfaces in $S$.

### *9.4. Collins' decomposition*

We define the *silhouette* [4, Section 5.3] of a surface defined by the equation $p(x, y, z) = 0$ by:

$$\begin{cases} p(x, y, z) = 0 \\ x\frac{\partial p}{\partial x} + y\frac{\partial p}{\partial y} + z\frac{\partial p}{\partial z} = 0. \end{cases}$$

The silhouette of a surface is the curve on the surface such that all points $(x, y, z)$ of the silhouette are such that the ray $[(x, y, z)]$ belongs to the tangent plane of the surface on $(x, y, z)$.

Up to infinitesimal perturbation of the coefficients of the polynomials, we can assume that the surfaces of $S$ have no integer points in $K$.

$\Pi(K) = \{\Pi(x) \mid x \in K\}$ is a compact of the affine plane $\mathcal{A}_1$. Let us consider the set $\Pi(S)$ of curves in $\Pi(K)$ containing:

- the projection of the silhouettes of surfaces in $S$;
- the projection of the intersections of surfaces in $S$ and of the intersection of surfaces in $S$ with the planes $z = \mu(1 + \frac{n}{6\delta})$, $n \in \{1, \ldots, 6\delta - 1\}$, where $\delta$ is the total degree of $S$;
- *vertical lines* of the form $\{(x, a, 1) \mid |x| \leq 2^{\beta+1}\}$ for $a$ a constant such that such lines pass through:
  - all intersections among the curves;

– all singular points of the curves;
– all critical points of the curves with a tangent supported by $\vec{e}_y$.

$\Pi(S)$ defines a Collins decomposition [40] of $\Pi(K)$. The intersection of any affine line supported by $\vec{e}_y$ of the plane with a region of this decomposition is connected if nonempty.

Let $c$ be a cell in $\Pi(S)$. It is enclosed by two curves in $\Pi(K)$ and at most two vertical lines. The curves can be parametrized by $c_{\max} : x \mapsto \max\{y \in \mathbf{R} \mid (x, y, 1) \in c\}$ and $c_{\min} : x \mapsto \min\{y \in \mathbf{R} \mid (x, y, 1) \in c\}$, which are both smooth functions. The *volatility* of $c$ is defined as the number of extrema of the second derivatives $c''_{\min}$ and $c''_{\max}$ on their domains of definition.

This set of curves $\Pi(S)$ can be lifted to a set of surfaces $\mathrm{Col}_S(K)$ of $K$ that contains:

- the surfaces of $S$;
- the cones $[s]$ of every curve $s$ in $\Pi(S)$;
- the planes bounding $K$;
- $6\delta - 2$ *dividing planes* of equation $z = \mu(1 + \frac{n}{6\delta})$, $n \in \{1, \ldots, 6\delta - 1\}$.

The projection of a cell of $\mathrm{Col}_S$ is a cell of $\Pi(S)$. We say that a cell of $\mathrm{Col}_S(K)$ is *flat* if none of its boundaries are included in surfaces of $S$.

Let us call $\mathbf{d}(S)$ the number of cells in $\mathrm{Col}_S(K)$.

Let $c$ be a cell in $\mathrm{Col}_S(K)$. Its *volatility* is defined as the volatility of its projection in $\Pi(S)$.

*9.5. Volatility and Separation*

We here follow but rephrase Mulmuley [4, Section 5.3; Sample points]

**Definition 64.** Let $K$ be a compact of $\mathbf{R}^3$.

A finite set of surfaces $S$ on $K$ *separates* a $\rho$-fan $\mathsf{Fan}$ on $K$ if the partition on $\mathbf{Z}^3 \cap K$ induced by $S$ is finer than the one induced by $\mathsf{Fan}$.

We now establish the following key result of Mulmuley [4, Theorem 5.9].

**Theorem 65.** *Let $S$ be a finite set of algebraic surfaces of total degree $\delta$, and $\mathsf{Fan}$ a $\rho$-fan of bitsize $\beta$.*

*If $S$ separates $\mathsf{Fan}$, there exists a compact $K$ and a cell of $\mathrm{Col}_S(K)$ with volatility greater than $\rho/\mathbf{d}(S)$.*

In order to prove this theorem, we will build explicitely the compact $K$ and this cell by considering sample points on $\mathsf{Fan}$ and show in Lemma 67 a bound on the volatility of this cell.

Let $K$ be a compact delimited by the cone of a rectangle $\{(x, y, 1) \mid |x|, |y| \leqslant 2^{\beta+1}\}$ and two planes of equation $z = \mu$ and $z = 2\mu$, with $\mu > (6\delta + 1)2^{\beta}$. We first remark that all affine segments of $\mathsf{Fan}$ are in the rectangle base of $K$.

For each affine segment of $\mathsf{Fan}$ with endpoints $(x_i, y_1, 1)$ and $(x_{i+1}, y_{i+1}, 1)$ let, for $0 < k < 10\mathbf{d}(S)$, $y_i^k$ be such that $(x_i^k, y_i^k, 1)$ is in the affine segment, where $x_i^k = \frac{(10\mathbf{d}(S)-k)x_i + kx_{i+1}}{10\mathbf{d}(S)}$. We remark that, as $|x_i - x_{i+1}| > 2^{-\beta}$, we have, for $k, k'$, $|x_i^k - x_i^{k'}| > 2^{-\beta}/10\mathbf{d}(S)$.

**Lemma 66.** *For all sample points $(x_i^k, y_i^k, 1)$, there exists a flat cell in* $\mathrm{Col}_S$ *that contains an integer point of* $[(x_i^k, y_i^k, 1)]$.

*Proof.* Let $(x_i^k, y_i^k, 1)$ be a sample point. $[(x_i^k, y_i^k, 1)]$ is divided in $N+1$ intervals by the dividing planes. On the other hand, $[(x_i^k, y_i^k, 1)]$ intersects surfaces of $S$ in at most $\delta$ points, by Bézout theorem. So, there exists an interval $e$ of $[(x_i^k, y_i^k, 1)]$ that is bounded by the dividing planes and that do not intersect any surface in $S$. By construction, $e$ is included in a flat cell, and its projection on the $z$-axis has length $\mu/(6\delta+1)$, so, as $(x_i^k, y_i^k, 1)$ is of bitsize $\beta$, $(n2^\beta x_i^k, n2^\beta y_i^k, n2^\beta)$ is, for all $n \in \mathbf{N}$ an integer point of the ray, so, as $\mu > (6\delta+1)2^\beta$, $e$ contains an integer point. □

So, for each affine segment of Fan, there exists a flat cell in $\mathrm{Col}_S$ that contains integer points in the ray of at least 10 sample points of the affine segment. Going further, there exists a cell $c$ of $\mathrm{Col}_S$ that contains integer points in the ray of at least 10 sample points of $\rho/\mathbf{d}(S)$ affine segments of Fan.

**Lemma 67.** *The volatility of $c$ is at least $\rho/\mathbf{d}(S)$.*

This is achieved by applying the mean value theorem on the function $\Pi(c)'_{\max}$ on pairs of sample points. In particular, this proof uses no algebraic geometry.

*Proof.* Let $e$ be a segment of Fan such that the ray of 10 of its sample points contain an integer point in $c$. Let $p = (x, y, z)$ be one of its integer point and $\Pi(p) = (x_p, y_p, 1)$ its projection, which is a sample point in $\Pi(c)$. Let $q = (x, y+1, z)$. As $\Pi(p)$ is in Fan, and $S$ separates Fan, $q$ is not in $c$, and $\Pi(q) = (x_q, y_q, 1)$ is not in $\Pi(c)$. By Thalès theorem, $0 < y_q - y_p < \frac{1}{\mu}$. So, as $y_q > \Pi(c)_{\max}(x_p) > y_p$, we have in particular that $0 < \Pi(c)_{\max}(x_p) - y_p < \frac{1}{\mu}$.

So, the 10 sample points have coordinates that approximate the graph of $\Pi(c)_{\max}$ with an error bounded by $\frac{1}{\mu}$. Consider two of them $p_1 = (x_1, y_1, 1)$ and $p_2 = (x_2, y_2, 1)$, such that $x_1 < x_2$. Let $a$ be the slope of $e$ (in particular $a = (y_2 - y_1)/(x_2 - x_1)$. By the mean value theorem, there exists $\alpha \in [x_1, x_2]$ such that $\Pi(c)'_{\max}(\alpha) = \frac{\Pi(c)_{\max}(x_2) - \Pi(c)_{\max}(x_1)}{x_2 - x_1}$. But $|\Pi(c)_{\max}(x_2) - \Pi(c)_{\max}(x_1)| \leq |y_2 - y_1| + \frac{2}{\mu}$ and $|x_2 - x_1| > \frac{1}{10\mathbf{d}(S)2^\beta}$. So, $|\Pi(c)'_{\max}(\alpha) - a| \leq 2\frac{10\mathbf{d}(S)2^\beta}{\mu}$.

So, the function $\Pi(c)'_{\max}$ is close to the value $a$, with error bounded, between all the sample points. By applying the mean value theorem again, we get that there exists a point in the interval such that $\Pi(c)''_{\max}$ is close to 0, with an error bounded by $2\frac{10\mathbf{d}(S)2^\beta}{\mu}$.

In the same way, let $e'$ be another segment of Fan such that the ray of 10 of its sample points contain an integer point in $c$, of slope $a'$. Let two of them be $p'_1 = (x'_1, y'_1, 1)$ and $p'_2 = (x'_2, y'_2, 1)$, and suppose $x'_2 > x'_1 > x_2$. By the same reasoning as above, there exists $\alpha' \in [x'_1, x'_2]$ such that $|\Pi(c)'_{\max}(\alpha') - a'| \leq 2\frac{10\mathbf{d}(S)2^\beta}{\mu}$. By the mean value theorem, there exists $\beta \in [\alpha, \alpha']$ such that $\Pi(c)''_{\max}(\beta) = \frac{\Pi(c)'_{\max}(\alpha') - \Pi(c)'_{\max}(\alpha)}{\alpha' - \alpha} > \frac{1}{\mu}(|a - a'| - 2\frac{10\mathbf{d}(S)2^\beta}{\mu})$.

So, for each of the $\rho/\mathbf{d}(S)$ segments of $\mathsf{Fan}$, we can exhibit a point such that $\Pi(c)''_{\max}$ is close to zero, and for each successive segment, a point such that it is far. So $\Pi(c)''_{\max}$ has at least $\rho/\mathbf{d}(S)$ extrema. □

*9.6. Volatility and Degree*

We can now state the following essential result.

**Theorem 68** (Mulmuley)**.** *Let $S$ be a finite set of algebraic surfaces of total degree $\delta$.*

*There exists a polynomial $P$ such that, for all $\rho > P(\delta)$, $S$ does not separate $\rho$-fans.*

While this Theorem underlies Mulmuley's proof technique, it is not explicitly stated in his article. This result follows from theorem 65 and the following two lemmas which appear as claims at the end of Section 5.3 in [4].

**Lemma 69.** *Let $S$ be a finite set of curves of total degree $\delta$, and $K$ be a compact. The cells of the decomposition $\mathrm{Col}_S$ of $K$ have a volatility bounded by a polynomial in $\delta$.*

*Proof.* Let $c$ be a cell in $\mathrm{Col}_S$ and $g(x, y) = 0$ be the equation of one of the boundaries of $\Pi(c)$ in the affine plane. The degree of $g$ is bounded by the degree of the intersection of surfaces in $S$. Any extrema $x$ of $f''$, where $f$ is a parametrization $y = f(x)$ of this boundary, can be represented as a point $(x, y, y^{(1)}, y^{(2)}, y^{(3)})$ in the 5-dimensional phase space that satisfy polynomial equations of the form:

$$g(x,y) = 0, \qquad g_1(x,y,y^{(1)}) = 0, \qquad g_2(x,y,y^{(1)},y^{(2)}) = 0$$
$$g_3(x,y,y^{(1)},y^{(2)},y^{(3)}) = 0, \qquad y^{(3)} = 0,$$

where all the polynomials' degrees are all bounded by the degree of the intersection of surfaces in $S$ (as they are the derivatives of $g$). So, by the Milnor–Thom theorem, such points are in number polynomial in the total degree of the surfaces of $S$. □

**Lemma 70.** *The number of cells $\mathbf{d}(S)$ of the Collins decomposition of $S$ is polynomial in $\delta$.*

*Proof.* The intersection of the surfaces in $S$ are algebraic varieties of number bounded by $\delta$, by the Milnor–Thom theorem. Moreover, so are the silhouettes of the surfaces, as they are the intersection of two algebraic varieties of total degree smaller than $\delta$. So, the number of cells in $\mathrm{Col}_S$ is bounded by the number of cells of $S$ times the number of dividing planes times the number of intersections, silhouettes and vertical lines they engender. □

## 10. Improving Mulmuley's result

*10.1.* PRAMs *over* $\mathbf{R}$ *and maxflow*

We will now prove our strengthening of Mulmuley's lower bounds for "PRAMs without bit operations" [4]. For this, we will combine the results from previous sections to establish the following result.

**Theorem 8.** *Let $N$ be a natural number and $M$ be a real-valued* PRAM *with at most $2^{O((\log N)^c)}$ processors, where $c$ is any positive integer.*

*Then $M$ does not decide* `maxflow` *on inputs of length $N$ in $O((\log N)^c)$ steps.*

*Proof of Theorem 8.* Let $N$ be an integer. Suppose that a real-valued PRAM $M$ with division and roots, with at most $p = 2^{O((\log N)^c)}$ processors, computes `maxflow` on inputs of length at most $N$ in time $k = 2^{O((\log N)^c)}$.

We know that $[\![M]\!]$ has a finite set of edges $E$. Since the running time of $M$ is equal, up to a constant, to the computation time of the $\text{CREW}^p(\alpha_{\mathbf{R}\text{full}})$-program $[\![M]\!]$, we deduce that if $M$ computes `maxflow` in $k$ steps, then $[\![M]\!]$ computes `maxflow` in at most $Ck$ steps where $C$ is a fixed constant.

By Lemma 55, the problem decided by $[\![M]\!]$ in $Ck$ steps defines a system of equations separating the integral inputs accepted by $M$ from the ones rejected. I.e. if $M$ computes `maxflow` in $Ck$ steps, then this system of equations defines a set of algebraic surfaces that separate the $\rho$-fan defined by `maxflow`. Moreover, this system of equation has a total degree bounded by $Ck \max(2, \sqrt[\partial]{G})2p \times 2^{O(\text{Card}(E))} \times 2^{k.h_0([\![\tilde{M}]\!])}$.

By Theorem 63 and Theorem 68, there exists a polynomial $P$ such that a finite set of algebraic surfaces of total degree $\delta$ cannot separate the $2^{\Omega(N)}$-fan defined by `maxflow` as long as $2^{\Omega(N)} > P(\delta)$. But here the entropy of $G$ is $O(p)$, as the entropy of a product $f \times g$ satisfies $h(f \times g) \leqslant h(f) + h(g)$ [37]. Hence $\delta = O(2^p 2^k)$, contradicting the hypotheses that $p = 2^{O((\log N)^c)}$ and $k = 2^{O((\log N)^c)}$. □

This extends Mulmuley's result because of the following fact.

**Proposition 71.** *A subset $A \subseteq \mathbf{Z}^k$ is decided by a division-free integer-valued* PRAMs *with $k$ processors in time $t$ if and only if there exists a division-free real-valued* PRAMs *with $k$ processors computing in time $t$ a subset $B \subseteq \mathbf{R}^k$ such that:*

$$\{(x_1, x_2, \ldots, x_k) \in \mathbf{Z}^k \mid (x_1, x_2, \ldots, x_k) \in B\} = A.$$

*Proof.* The proof is rather straightforward, since addition and multiplication of integers yield integers. As a consequence, the available operations in division-free real PRAMs cannot be used to construct non-integer values from solely integer inputs. □

A consequence of the previous result is that Mulmuley's original result is obtained as a corollary of theorem 8. Indeed, suppose a PRAM without bit operations computes the `maxflow` problem in polylogarithmic time. Then there

would exist a real-valued PRAM computing `maxflow` in polylogarithmic time, a result contradicting Theorem 8.

Let us now consider the possibility of lifting this result to integer-valued machines using division. Let $M$ be an integer-valued PRAM. We would like to associate to it a real-valued PRAM $\widetilde{M}$ such that $M$ and $\widetilde{M}$ accept the same (integer) values, with at most a polylogarithmic running time overhead. This implies in particular that real-valued PRAMs (with division, and potentially roots) should be able to compute euclidean division efficiently. It turns out that this is not the case. Indeed, we will show that the remainder of the euclidean division by 2 is in fact not computable in polylogarithmic time by real-valued PRAMs even in the presence of division and arbitrary root operations. This will be obtained using the above results based on entropic co-trees and Mumuley's geometric argument. However, before providing the proof of this result, we provide a more concrete version of a similar result on algebraic circuits. This proof illustrates in a simpler setting the abstract techniques we developed above.

*10.2. Real* PRAMs *and euclidean division*

We now give a direct proof that algebraic circuits cannot compute the parity function. More precisely, we consider the modulo function (noted %) over $[\![0; 2^n]\!]$. We prove that the function $(n \mapsto n\%2)$ cannot be computed by a family of real circuits of ALGCIRC of largeness $\text{poly}(n)$ and of depth $\text{polylog}(n)$ over integers of $[\![0; 2^n]\!]$. In our proof, we consider that comparison gates returns 1 if true, 0 otherwise. This will ease the proof, while being equivalent to Definition 20.

In this section we do a direct analysis of circuits of ALGCIRC (circuits over reals with division gate, but no root gates). We will prove that the modulo function, cannot be computed by such a circuit when we restrict inputs to integers. One objection quoted by Mulmuley in his paper [4] is that proving computing the parity function cannot be done with "PRAM without bit operation", is akin to proving that non monotone functions cannot be computed by monotone circuits. We disagree to the extent that proving that euclidian division is not computed by a circuit of ALGCIRC is not trivial like in the monotone function case and actually encapsulates the reason why algebraic circuits of low depth are fundamentally limited. The core of our argument is the same as the one Mulmuley gives in his paper, however we feel that our formalism may help in understanding what are PRAMs of polylog depth and why they are weak. On top of his comparison with monotone functions Mulmuley adds that anyhow "a lower bound in the PRAM model without bit operations is interesting only if the problem has an efficient sequential algorithm that does not use bit operations [...] One trivial example of a problem that does not have a strongly polynomial time algorithm is bit extraction itself (because such an algorithm for bit extraction must work within O(1) arithmetic and comparison operations, which is impossible)". We then give an example of a very simple problem which has a strongly polynomial time algorithm but is not computed by algebraic circuits: deciding the set

$$\left\{ (x_1, \ldots, x_n \in \mathbf{N}); (\sum_{i=1}^{n} x_i)\%2 = 0 \right\}.$$

A similar argument can also be used to prove that the natural bijection from $\mathbf{N}$ to $\mathbf{N}^2$ cannot be computed by algebraic circuits (and therefore by PRAMs without bit operations in polylogarithmic time), which is an interesting limitation to notice.

The main idea of the proof is a more concrete statement of the technical lemma above: we show that circuits of ALGCIRC compute piece-wise polynomial fractions with few zeroes and few *pieces* and argue that it therefore can't compute the euclidean division.

**Definition 72.** A set $I$ over $\mathbf{R}$ is an interval if there exists two real numbers $a, b \in \mathbf{R} \cup \{-\infty; +\infty\}$ such that either $I = [a, b]$, or $I = ]a, b]$, or $I = [a, b[$, or $I = ]a, b[$.

**Definition 73** (Comparison of intervals)**.** Let $I$ and $J$ be two non empty, non intersecting intervals over $\mathbf{R}$. We say $I < J$ if $\forall x \in I, \forall y \in J, x < y$.

Given a collection of $r > 0$ intervals $(I_i)_{i=0}^{r}$, we write $I_1 < I_2 < \ldots < I_r$ to indicate the family is ordered and all the intervals are empty and non intersecting.

Note that what we call here "pieces" will play the role of (and are in fact) the cells in the $k$-cell decomposition exposed above. In particular, the main technical result leading to the lower bound is an upper bound on the number of pieces and on the *extended degree* of the piece-wise rational functions defined by a circuit of a given depth. This should be understood as a special case of our general technical lemma (Lemma 5) giving bounds on the number of cells and the number and degrees of the polynomials defining each cell (here the notion of extended degree captures both the number of equations and their degree, which is possible because we work in a simpler setting).

**Definition 74** (Interval cut and pieces)**.** Given a family of $r$ intervals $I_1 < I_2 < \ldots < I_r$ $(r > 0)$, we say it is a *size $r$ interval cut* if $I_1, I_2, \ldots, I_r$ partitions $\mathbf{R}$. Each $I_i$ is called a *piece*.

In the following we may use as a shorthand the notation $I$ for a family of $r$ intervals $(I_i)_{1 \leq i \leq r}$.

**Definition 75** (Intertwining interval cuts)**.** Suppose given a size $n$ interval cut $I_1 < I_2 < \ldots < I_n$ and a size $m$ interval cut $J_1 < J_2 < \ldots < J_m$. There is at least one size $k$ interval cut $K_1, \ldots, K_k$ such that $\forall h, \exists i, j, K_h \subset I_i \wedge K_h \subset J_j$. If $k$ is moreover minimal then we call $K_1, \ldots, K_k$ an *intertwining* of $I$ and $J$.

As an example of intertwining consider the two families $I = (]-\infty; 0], ]0; +\infty[)$ and $J = (]-\infty; 1], ]1; +\infty[)$; one possible intertwining is $(]-\infty; 0], ]0; 1], ]1; +\infty[)$. Note that the size $k$ of an intertwining between size $n$ and size $m$ interval cuts satisfies $k \leqslant n + m$ (consider the extremal points of each interval).

**Definition 76** (Piece-wise polynomial fraction)**.** Consider given a size $n$ interval cut $I_1 < I_2 < \ldots < I_n$. A function $f$ over $D \subset \mathbf{R}$ is said to be a piece-wise polynomial fraction over $I$ if there exists $n$ polynomial fractions $f_1, \ldots, f_n$ such that for all integer $1 \leqslant i \leqslant n$: $\forall x \in I_i \cap D, f(x) = f_i(x)$.

The number of pieces of a piece-wise polynomial fraction $f$ is the smallest number $n$ such that $f$ is a polynomial fraction over a size $n$ interval cut. We write $c$ the function which associates to each piece-wise polynomial fraction $f$ and return its minimal number of pieces. In the following, we will abusively apply $c$ to circuits that compute piece-wise polynomial fractions.

Let us note that the poles of a polynomial fraction do not requires a splitting in pieces. For instance, the function $f(x) = \frac{1}{x^2-1}$ defined for all $x \in \mathbf{R} \setminus \{-1; 1\}$ is a one piece polynomial fraction. If one wanted to have functions defined over the whole of $\mathbf{R}$ it would be no issue as whenever performing a division we could add a comparison gate in the circuit to verify that we are not dividing by 0, and if so return an arbitrary value.

Let $F$ be a piece-wise polynomial fraction over $D \subset \mathbf{R}$, there can be multiple interval cut associated to the number of pieces of $F$, for instance the function which is 0 for negative numbers and $x \mapsto x$ for non negative numbers as two minimal cuts : $(]-\infty; 0], ]0; +\infty[)$ or $(]-\infty; 0[, [0; +\infty[)$. So in the following we will speak of *an* interval cut of $F$. Once we have fixed an interval cut $I$, the associated polynomial functions are unique except on intervals $I_i$ such that $I_i \cap D = \{a\}$ is a singleton, for those intervals we always associate the polynomial of degree 0, $f_i = (x \mapsto f(a))$. This is important for the following definition.

**Definition 77** (Augmented degree of a polynomial fraction)**.** Let $P$ and $Q$ be polynomials over $\mathbf{R}$. We define the *augmented degree* $d(F)$ of the polynomial fraction $F = \frac{P}{Q}$ as $d(F) = d(P) + d(Q) + 1$, where $d(P), d(Q)$ are the degrees of the polynomials $P, Q$ in the standard meaning.

Let $f$ be a piece-wise polynomial fraction over $\mathbf{R}$ with a minimal interval cut $I_1 < \ldots < I_n$ and $f_1, \ldots, f_n$ the associated polynomial fractions. We define the extended degree of $f$ by $d(f) = max_{1 \leq i \leq n} d(f_i)$. In the following we will abusively apply $d$ to circuits that compute piece-wise polynomial fractions.

**Theorem 78.** *Let $P$ be a real circuit of depth $k$ taking only one input. Then $P$ computes a piece-wise polynomial fraction such that $d(P) \leq 2^k$ and $c(P) \leq 2^{\frac{k^2+3k}{2}}$.*

*Proof.* We prove the result for each gate by induction on the depth of the gate. I.e. we prove that the circuit computes a piece-wise polynomial fraction, and that the stated bounds are correct. We denote by $c_k$ (resp. $d_k$) the maximal number of pieces (resp. the maximal augmented degree) of a function computed by a gate at depth $k$. Let $p$ be a gate, $c(p)$ corresponds to the augmented degree of the function computed by $p$, $d$ to its augmented degree.

For depth $k = 1$ the function computed is either $f = (x \mapsto x)$ or $g = (x \mapsto 1)$. Notice that $d(f) = 2$, $c(f) = 1$, $d(g) = 1$ and $c(g) = 1$. Therefore $d_1 \leq 2^1$ and $c_1 \leq 2^2$.

For the induction step, let us consider a gate called $p$ at depth $k + 1$. Our induction hypothesis is that $d_k \leq 2^k$ and $c_k \leq 2^{\frac{k(k+3)}{2}}$. We now consider the different possible gates $p$.

- If $p$ is a multiplication gate: $p = p_1 * p_2$. Let $I_1 < \ldots < I_r$ and $J_1, < \ldots < J_m$ be the minimal size interval cuts of $p_1$ and $p_2$, and $r = c(p_1)$ and $m = c(p_2)$. Let us consider $K_1, \ldots, K_k$ an intertwining of $I$ and $J$ such that $k \leqslant r + m$ . We note that $p$ is a piece-wise polynomial fraction over $K$. Therefore $c(p) \leq c(p_1) + c(p_2) \leq 2 * c_k \Rightarrow c_(p) \leq 2 * c_k \leq 2^{\frac{(k+1)(k+4)}{2}}$ . By reasoning over each piece of the interval cut $K$ we also obtain that $d(p) \leq d(p_1) + d(p_2) \leq 2d_k$.
- If $p$ is either an addition or a division gate, the reasoning is similar.
- If $p$ is a comparison gate: $p = (p_1 \leq p_2)$. Let $A_1 \leq A_2 \leq \ldots \leq A_n$ be the pieces of $p_1$, $B_1 \leq B_2 \leq \ldots \leq B_m$ be the pieces of $p_2$, and define $n = c(p_1)$ and $m = c(p_2)$. Let $I_1 \leq I_2 \leq \ldots \leq I_r$ be an intertwining of $A$ and $B$. For all integer $1 \leqslant i \leqslant r$, $p_1$ and $p_2$ are both polynomial fraction over $I_i$ and we can canonically write $p_{i,1} = \frac{q_{i,1}}{t_{i,1}}$ and $p_{i,2} = \frac{q_{i,2}}{t_{i,2}}$. To count the number of pieces of $p$, we focus on each $I_i$ separately. One can notice that over $I_i$ any new piece of $p$ contained in $I_i$ may only occur on places where $p_{i,1} - p_{i,2}$ changes sign, moreover $p_{i,1} \leq p_{i,2}$ if and only if either $q_{i,1} * t_{i,2} - t_{i,1} * q_{i,1} \leq 0$ or $q_{i,1} * t_{i,2} - t_{i,1} * q_{i,1} \geq 0$ (depending on the signs of the denominators). But the polynomial $q_{i,1} * t_{i,2} - t_{i,1} * q_{i,1}$ has at most $d(p_{i,1}) + d(p_{i,2}) - 1$ roots, therefore over $I_i$ we "add" at most $d(p_{i,1}) + d(p_{i,2})$ pieces. Since we have at most $r \leq n + m = c(p_1) + c(p_2)$ pieces $I_i$, we have that $c(p)$ is at most $(d(p_1)+d(p_2)) * (c(p_1)+c(p_2)) \leq 2d_k * 2c_k \leq 2 * 2^k * 2 * 2^{\frac{k^2+3k}{2}} \leq 2^{\frac{(k+1)^2+3(k+1)}{2}}$. And $d(p)$ is equal to $1 \leq 2^{k+1}$.

□

The next lemma state that no piece-wise polynomial fraction with few pieces or low augmented degree can agree with the remainder function on many consecutive integers.

**Lemma 79.** *Let $f$ be a piece-wise polynomial fraction over $\mathbf{R}$, $N$ an integer. If $\forall k \in [\![0; N]\!], f(k) = k\%2$, then $d(f) * c(f) \geq \frac{N}{10}$.*

*Proof.* Let $f$ be a piece-wise polynomial fraction as described in the theorem, and $c$ its number of pieces with $I_1, \ldots, I_c$ a corresponding interval cut. If $c$ is less than $N/10$, there must be a piece $I_j$ containing at least $N/c$ integers of $[\![0; N]\!]$. We remind that over $I_j$, the function $f$ is a polynomial fraction which we note $\frac{P}{Q}$. Since $k \mapsto k\%2$ has $N/2c$ zeroes over $I_j$, so must $P$. The polynomial $P$ also cannot be the zero function because $k \mapsto k\%2$ is equal to 1 for some integers in $I_j$. Therefore P is of degree at least $N/2c$, and $d(f) * c \geq (N/2c) * c \geq N/10$. □

**Theorem 80.** *For any function family $(f_n)$ computed by a real-valued circuit family of depth $c \log^d(n)$, there exists $N \in \mathbf{N}$ such that $\forall n > N, \exists k \in [\![0; 2^n]\!], f_n(k) \neq k\%2$.*

*Proof.* Let $(C_n)_n$ be a sequence of circuits of depth $k_n = O(c \log^d(n))$. Since $n \mapsto n\%2$ is a function with one input, we may consider $(C_n)_n$ to be a sequence of circuits with one input. For any $n$, the function computed by $C_n$ is a piece-wise polynomial fraction $f$. By theorem 78 we have that $c(f) \leq 2^{\frac{k_n^2+3k_n}{2}}$ and

$d(f) \leq 2^{k_n}$ therefore $c(f)d(f) = o(2^n)$. Therefore for large enough $n$, using lemma 79, $f$ cannot coincide with $n \mapsto n\%2$ for all integers in $[\![0; 2^n]\!]$. □

We continue by proving that the two problems given in introduction are indeed not computable by algebraic circuits.

**Theorem 81.** *The set*

$$\left\{ (x_1, \ldots, x_n \in \mathbf{N}); (\sum_{i=1}^{n} x_i)\%2 = 0 \right\}.$$

*has polynomially bounded arithmetic circuits but is not computable by polylogarithmic depth algebraic circuits.*

*Proof.* One can retrieve with a circuit of size linear in $n$ the last bit of $\sum_{i=1}^{n} x_i$. One the other hand the set

$$\left\{ (0, \ldots, 0, x_n \in \mathbf{N}); (\sum_{i=1}^{n} x_i)\%2 = 0 \right\}.$$

is not computable by polylogarithmic depth algebraic circuits by Theorem 80. □

**Theorem 82.** *Let $f : N \mapsto N \times N$ be the usual Cantor pairing function $f(n, m) = \frac{1}{2}(m+n)(m+n+1)+m$. This function is bijective and its reciprocal $f^{-1}$ cannot be computed by a polylogarithmic depth algebraic circuit.*

*Proof.* Consider the function $g(x, y) = (x < y)$. (it returns either 0 or 1) The function $g(f^{-1})$ has $2^{\frac{n}{2}}$ alternations between 0 and 1 over $[\![0; 2^n]\!]$ therefore it cannot be computed by a polylogarithmic depth algebraic circuit (by arguments used to prove theorem 79 and 80 . But $g$ can be computed by a polylogarithmic depth algebraic circuit. As a consequence, it must be $f^{-1}$ which cannot be computed by a polylogarithmic depth algebraic circuit. □

We haven't been able to use this analysis to include circuits with square root gates. Everything may still hold, but we haven't been able to prove satisfactory bounds on the number of zeros of function computed by circuits with square root gates. This is important because when considering the comparison gates, the number of pieces created directly depends on the number of zeroes of the function.

So, while the arguments of this section show a more concrete application of the ideas behind the method used in this paper, this version is more limited. In the next section, we will extend this result to the model with arbitrary roots using the more abstract method developed in previous sections.

### *10.3. Extending to* PRAMs *with roots*

The above theorem is a particular case of the following result, which we prove using the general technique developed in the previous sections. We note that we

do not know of a more concrete proof using arguments similar to the ones used in the previous section.

Using the general bounds provided by entropic co-trees (Lemma 55) and the geometric result extracted from Mulmuley's geometric proof of lower bounds (Theorem 68), we show that euclidean division by 2 cannot be computed by real-valued PRAMs (with division and arbitrary root operations) on inputs of length $N$ in polylogarithmic time in $N$.

**Theorem 1.** *Let $N$ be a natural number and $M$ be a real-valued* PRAM *with at most $2^{O((\log N)^c)}$ processors, where $c$ is any positive integer.*

*Then $M$ does not compute euclidean division by 2 on inputs of length $N$ in $O((\log N)^c)$ steps.*

*Proof.* Suppose that a real-valued PRAM $M$ with division and roots, with at most $p = 2^{O((\log N)^c)}$ processors, computes euclidean division by 2 on inputs of length at most $N$ in time $k = 2^{O((\log N)^c)}$. We know that $[\![M]\!]$ has a finite set of edges $E$, and the running time of $M$ is equal, up to a constant, to the computation time of the CREW$^p(\alpha_{\mathbf{R}\text{full}})$-program $[\![M]\!]$, we deduce that if $M$ computes euclidean division by 2 in $k$ steps, then $[\![M]\!]$ computes euclidean division by 2 in at most $Ck$ steps where $C$ is a fixed constant.

Consider the following optimisation problem:

$$\{(x, y, 1) \mid y \leqslant x//2\}$$

It is defined by the frontier set

$$\mathsf{Front} = \{(x, y, 1) \in \mathbf{R}^3 \mid y = x//2\}.$$

and we remark that the induced cone

$$[\mathsf{Front}] = \{(x, y, z) \in \mathbf{R}^2 \times \mathbf{R}^+ \mid y/z = \mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(x/z)\}.$$

is a $\rho$-fan where $\rho = 2^{\Omega(N)}$ is exponential in the maximal size of the inputs.

By Lemma 55, the problem decided by $[\![M]\!]$ in $Ck$ steps defines a system of equations separating the integral inputs accepted by $M$ from the ones rejected. I.e. if $M$ computes euclidean division by 2 in $Ck$ steps, then this system of equations defines a set of algebraic surfaces that separate the $\rho$-fan defined above. Moreover, this system of equation has a total degree bounded by $Ck \max(2, \sqrt[\partial]{G})2p \times 2^{O(\mathrm{Card}(E))} \times 2^{k.h_0([\![\tilde{M}]\!])}$.

Now by Theorem 68, there exists a polynomial $P$ such that a finite set of algebraic surfaces of total degree $\delta$ cannot separate the $2^{\Omega(N)}$-fan defined by euclidean division by 2 as long as $2^{\Omega(N)} > P(\delta)$. But here the entropy of $G$ is $O(p)$ from[17] lemma 43. Hence $\delta = O(2^p 2^k)$, contradicting the hypotheses that $p = 2^{O((\log N)^c)}$ and $k = 2^{O((\log N)^c)}$. □

[17] Once again using the fact that $\alpha_{\mathbf{R}\text{full}}$ possesses the property that non-central elements act as the identity on the private memory.

## References


[1] J. M. Steele, A. Yao, Lower bounds for algebraic decision trees, Journal of Algorithms 3 (1982) 1–8. doi:10.1016/0196-6774(82)90002-5.

[2] M. Ben-Or, Lower bounds for algebraic computation trees, in: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, STOC '83, ACM, New York, NY, USA, 1983, pp. 80–86. doi:10.1145/800061.808735.
URL http://doi.acm.org/10.1145/800061.808735

[3] F. Cucker, $\mathbf{P_r} \neq \mathbf{NC_r}$, Journal of Complexity 8 (3) (1992) 230 – 238. doi:https://doi.org/10.1016/0885-064X(92)90024-6.
URL http://www.sciencedirect.com/science/article/pii/0885064X92900246

[4] K. Mulmuley, Lower bounds in a parallel model without bit operations, SIAM J. Comput. 28 (4) (1999) 1460–1509. doi:10.1137/S0097539794282930.
URL https://doi.org/10.1137/S0097539794282930

[5] A. Cobham, The intrinsic computational difficulty of functions, in: Proceedings of the 1964 CLMPS, 1965.

[6] J. Edmonds, Paths, trees and flowers, Canad. J. Math. 17 (1965) 449–467.

[7] M. O. Rabin, Mathematical theory of automata, Proc. Symp. Appl. Math. AMS 19 (1967) 159–160.

[8] S. Cook, The complexity of theorem-proving procedures, in: Proceedings of the 3rd ACM Symposium on Theory of Computing, 1971.

[9] R. Williams, Nonuniform acc circuit lower bounds, J. ACM 61 (1) (2014) 2:1–2:32. doi:10.1145/2559903.
URL http://doi.acm.org/10.1145/2559903

[10] T. Baker, J. Gill, R. Solovay, Relativizations of the p = np question, SIAM Journal on Computing 4 (4) (1975) 431–442. doi:10.1137/0204037.

[11] A. A. Razborov, S. Rudich, Natural proofs, Journal of Computer and System Sciences 55 (1) (1997) 24 – 35. doi:https://doi.org/10.1006/jcss.1997.1494.

[12] S. Aaronson, A. Wigderson, Algebrization: A new barrier in complexity theory, ACM Trans. Comput. Theory 1 (1) (2009) 2:1–2:54. doi:10.1145/1490270.1490272.
URL http://doi.acm.org/10.1145/1490270.1490272

[13] B. Aydinlioğlu, E. Bach, Affine relativization: Unifying the algebrization and relativization barriers, ACM Trans. Comput. Theory 10 (1) (2018). doi:10.1145/3170704.

[14] K. D. Mulmuley, The gct program toward the p vs. np problem, Commun. ACM 55 (6) (2012) 98–107. doi:10.1145/2184319.2184341.
URL http://doi.acm.org/10.1145/2184319.2184341

[15] L. Fortnow, The status of the p versus np problem, Commun. ACM 52 (9) (2009) 78–86. doi:10.1145/1562164.1562186.
URL http://doi.acm.org/10.1145/1562164.1562186

[16] P. Bürgisser, C. Ikenmeyer, G. Panova, No occurrence obstructions in geometric complexity theory, in: 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), 2016, pp. 386–395. doi:10.1109/FOCS.2016.49.

[17] C. Ikenmeyer, U. Kandasamy, Implementing geometric complexity theory: On the separation of orbit closures via symmetries, in: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Association for Computing Machinery, New York, NY, USA, 2020, p. 713–726. doi:10.1145/3357713.3384257.
URL https://doi.org/10.1145/3357713.3384257

[18] C. A. Neff, Specified precision polynomial root isolation is in NC, Journal of Computer and System Sciences 48 (3) (1994) 429 – 463. doi:https://doi.org/10.1016/S0022-0000(05)80061-3.

[19] L. R. Ford, D. R. Fulkerson, A simple algorithm for finding maximal network flows and an application to the hitchcock problem, Canadian Journal of Mathematics (1957) 210–218.

[20] L. M. Goldschlager, R. A. Shaw, J. Staples, The maximum flow problem is log space complete for p, Theoretical Computer Science 21 (1982) 105–111. doi:https://doi.org/10.1016/0304-3975(82)90092-5.

[21] T. Seiller, Interaction graphs: Full linear logic, in: IEEE/ACM Logic in Computer Science (LICS), 2016.
URL http://arxiv.org/pdf/1504.04152

[22] T. Seiller, Interaction graphs: Graphings, Annals of Pure and Applied Logic 168 (2) (2017) 278–320. doi:10.1016/j.apal.2016.10.007.

[23] T. Seiller, Interaction graphs: Nondeterministic automata, ACM Transaction in Computational Logic 19 (3) (2018).

[24] T. Seiller, Interaction Graphs: Exponentials, Logical Methods in Computer Science Volume 15, Issue 3 (Aug. 2019). doi:10.23638/LMCS-15(3:25)2019.
URL https://lmcs.episciences.org/5730

[25] J. Milnor, On the Betti numbers of real varieties, in: Proceedings of the American Mathematical Society, 1964, p. 275. doi:10.2307/2034050.

[26] O. Oleĭnik, I. Petrovskii, On the topology of real algebraic surfaces, Izv. Akad. Nauk SSSR 13 (1949) 389–402.

[27] R. Thom, Sur l'homologie des variétés algébriques réelles, Princeton University Press, 1965, pp. 255–265.

[28] S. Adams, Trees and amenable equivalence relations, Ergodic Theory and Dynamical Systems 10 (1990) 1–14.

[29] D. Gaboriau, Coût des relations d'équivalence et des groupes, Inventiones Mathematicae 139 (2000) 41–98. doi:10.1007/s002229900019.

[30] D. Gaboriau, Invariants $\ell^2$ de relations d'équivalence et de groupes, Publ. Math. Inst. Hautes Études Sci 95 (93-150) (2002) 15–28.

[31] T. Seiller, Zeta functions and the (linear) logic of markov processes, https://hal.archives-ouvertes.fr/hal-02458330 (2021).

[32] T. Seiller, Towards a *Complexity-through-Realizability* theory, http://arxiv.org/pdf/1502.01257 (2015).

[33] P. J. Carstensen, The complexity of some problems in parametric linear and combinatorial programming, Ph.D. thesis, Ann Arbor, MI, USA (1983).

[34] A. C.-C. Yao, Decision tree complexity and betti numbers, Journal of Computer and System Sciences 55 (1) (1997) 36 – 43. doi:https://doi.org/10.1006/jcss.1997.1495.

[35] R. L. Adler, A. G. Konheim, M. H. McAndrew, Topological entropy, Transactions of the American Mathematical Society 114 (2) (1965) 309–319.

[36] J. E. Hofer, Topological entropy for noncompact spaces, The Michigan Mathematical Journal 21 (3) (1975) 235–242. doi:10.1307/mmj/1029001311.

[37] L. W. Goodwyn, The product theorem for topological entropy, Transactions of the American Mathematical Society 158 (2) (1971) 445–452.
URL http://www.jstor.org/stable/1995916

[38] L. Blum, M. Shub, S. Smale, On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, American Mathematical Society. Bulletin. New Series 21 (1) (1989) 1–46. doi:10.1090/S0273-0979-1989-15750-9.

[39] K. G. Murty, Computational complexity of parametric linear programming, Mathematical programming 19 (1) (1980) 213–219.

[40] G. E. Collins, Quantifier elimination for real closed fields by cylindrical algebraic decomposition, in: Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20–23, 1975, Springer, 1975, pp. 134–183.